阿里云
轻量级分布式应用服务

应用开发

文档版本：20190831

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 Spring Cloud 开发

1.1 Spring Cloud 开发概述

SAE 支持原生 Spring Cloud 微服务框架，在该框架下开发的应用只需添加依赖和修改配置，即可获取 SAE 企业级的应用托管、应用治理、监控报警和应用诊断等能力，实现代码零代码量应用迁移。

Spring Cloud 提供了简化应用开发的一系列标准和规范。这些标准和规范包含了服务发现、负载均衡、熔断、配置管理、消息事件驱动、消息总线等，同时 Spring Cloud 还在这些规范的基础上，提供了服务网关、全链路跟踪、安全、分布式任务调度和分布式任务协调的实现。

目前业界比较流行的 Spring Cloud 具体实现有 Spring Cloud Netflix、Spring Cloud Consul 、Spring Cloud Gateway、Spring Cloud Sleuth 等，最近由阿里巴巴中间件开源的 Spring Cloud Alibaba 也是业界中受关注度很高的另一种实现。

如果您已经使用 Spring Cloud Netflix、Spring Cloud Consul 等 Spring Cloud 组件开发的应用，可以直接部署到 SAE 正常运行并获得应用托管能力，同时还可以不修改任何一行代码直接使用 SAE 所提供的高级监控功能，实现全链路跟踪、监控报警和应用诊断等监控功能。

如果您的 Spring Cloud 应用想使用 SAE 中更多的服务治理相关的功能，那么您需要将您的 Spring Cloud 组件替换为 Spring Cloud Alibaba 中的组件或增加 Spring Cloud Alibaba 组件。

兼容性说明

SAE 目前支持 Spring Cloud Greenwich、Spring Cloud Finchley 和 Spring Cloud Edgware 三个版本。Spring Cloud、Spring Boot 和 Spring Cloud Alibaba 及各组件的版本对应关系请参见版本配套关系说明。

Spring Cloud 功能 、开源实现及 SAE 兼容性如下表所示：

Spring Cloud 功能		开源实现	SAE 兼容性
通用功能	服务注册与发现	· Netflix Eureka · Consul Discovery	兼容且提供替换组件
	负载均衡	Netflix Ribbon	兼容
	服务调用	· Feign · RestTemplate	兼容

Spring Cloud 功能	开源实现	SAE 兼容性
配置管理	· Config Server · Consul Config	兼容且提供替换组件
服务网关	· Spring Cloud Gateway · Netflix Zuul	兼容
链路跟踪	Spring Cloud Sleuth	兼容且提供替换组件
消息驱动 Spring Cloud Stream	· RabbitMQ binder · Kafka binder	兼容且提供替换组件
消息总线 Spring Cloud Bus	· RabbitMQ · Kafka	兼容且提供替换组件
安全	Spring Cloud Security	兼容
分布式任务调度	Spring Cloud Task	兼容
分布式协调	Spring Cloud Cluster	兼容

版本配套关系说明

Spring Cloud、Spring Boot 和 Spring Cloud Alibaba 及 SAE 提供的商业化组件的版本配套关系如下表所示。

Spring Cloud	Spring Boot	Spring Cloud Alibaba	
ANS	ACM	SchedulerX	
Greenwich	2.1.x	0.9.0.RELEASE	
Finchley	2.0.x	0.2.2.RELEASE	
Edgware	1.5.x	0.1.2.RELEASE	

说明:

Spring Cloud Alibaba Nacos Discovery 和 Spring Cloud Alibaba Nacos Config 分别 ANS 和 ACM 对应的开源组件。

1.2 初次使用 Spring Cloud 部署微服务应用

初次使用原生 Spring Cloud 部署微服务应用时，您需在本地程序中添加依赖配置实现服务注册与发现，并在本地测试调用结果后，将服务提供者和消费者分别部署到 SAE。

- 下载 Maven 并设置环境变量。
- 下载最新版本的 Nacos Server，并按以下步骤启动 Nacos Server。
  1. 解压下载的 Nacos Server 压缩包。
  2. 进入`nacos/bin`目录，执行如下命令启动 Nacos Server。
     - Linux/Unix/Mac 系统：执行命令`sh startup.sh -m standalone`。
     - Windows 系统：双击执行`startup.cmd`文件。

步骤一：获取 Demo

- service-provider 下载>>
- service-consumer 下载>>

步骤二：创建服务提供者

在本地创建命名为`nacos-service-provider`的Spring Cloud 工程，修改 pom 依赖，开启服务注册与发现，并将注册中心指定为 Nacos Server。

1. 创建名为 *nacos-service-provider* 的 Maven 工程。
2. 添加 pom 依赖。

   在 `pom.xml` 文件中添加依赖：

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
        <relativePath/>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-discovery
</artifactId>
            <version>0.9.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
```

```
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Greenwich.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

Spring Cloud Alibaba 版本说明：

- **示例中使用的版本为 Spring Cloud Greenwich，对应 Spring Cloud Alibaba 版本为 0.9.0.RELEASE。**
- **如果使用 Spring Cloud Finchley 版本，对应 Spring Cloud Alibaba 版本为 0.2.2.RELEASE。**
- **如果使用 Spring Cloud Edgware 版本，对应 Spring Cloud Alibaba 版本为 0.1.2.RELEASE。（Spring Cloud Edgware 版本的生命周期即将在 2019 年 8 月结束，不推荐使用这个版本开发应用。）**

3. **开启服务注册与发现。**

   a. **在`src\main\java`下创建命名为`com.aliware.sae`的 Package。**

   b. **在 Package`com.aliware.sae`中创建服务提供者的启动类`ProviderApplication`，并添加以下代码，其中`@EnableDiscoveryClient`注解表明此应用需开启服务注册与发现功能。**

```
    package com.aliware.edas;

    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBoot
Application;
    import org.springframework.cloud.client.discovery.EnableDisc
overyClient;

    @SpringBootApplication
    @EnableDiscoveryClient
    public class ProviderApplication {

        public static void main(String[] args) {
            SpringApplication.run(ProviderApplication.class, args
);
        }
    }
```

4. 提供 Echo 服务。

   在 Package`com.aliware.edas`中创建`EchoController`，指定 URL mapping 为 *`{/echo/{String}}`*，指定 HTTP 方法为 GET，方法参数从 URL 路径中获得，回显收到的参数。

```
package com.aliware.edas;

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EchoController {
    @RequestMapping(value = "/echo/{string}", method = RequestMethod
.GET)
    public String echo(@PathVariable String string) {
        return string;
    }
}
```

5. 修改配置。

   在`src\main\resources`路径下创建文件`application.properties`，并在该文件中添加以下配置，指定 Nacos Server 的地址。

```
xml
spring.application.name=service-provider
server.port=18081
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

   其中`127.0.0.1`为 Nacos Server 的IP地址。如果您的 Nacos Server 部署在另外一台机器，则需要修改成对应的 IP 地址。如果有其它需求，可以参考配置项参考在`application.properties`文件中增加配置。

6. 查询应用服务。

   a. 执行`nacos-service-provider`中`ProviderApplication`的`main`函数，启动应用。

   b. 登录本地启动的 Nacos Server 控制台`http://127.0.0.1:8848/nacos`（本地 Nacos 控制台的默认用户名和密码均为 `nacos`），在左侧导航栏中选择服务管理 > 服务列表，可以看到服务列表中已经包含了`service-provider`，且在详情中可以查询该服务的详情。

步骤三：创建服务消费者

在本地创建名为`nacos-service-consumer` Spring Cloud 工程。修改 pom 依赖，开启服务注册与发现，将注册中心指定为 Nacos Server。添加配置，使消费者在 RestTemplate 和 FeignClient 这两个客户端去调用服务提供者。

1. 创建 nacos-service-consumer 工程命名为`nacos-service-consumer`的 Maven 工程。
2. 添加 pom 依赖。

   在`pom.xml`文件中添加以下依赖。

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
        <relativePath/>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-discovery
</artifactId>
            <version>0.9.0.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Greenwich.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

3. 开启服务注册与发现。

   与服务提供者 `nacos-service-provider` 相比，除了开启服务注册与发现外，还需要添加两项配置才能使用 RestTemplate 和 FeignClient 这两个客户端：

   a. 在`src\main\java`下创建命名为 `com.aliware.sae`的Package。

   b. 在 Package`com.aliware.sae`中配置 RestTemplate 和 FeignClient。

      A. 在 Package`com.aliware.edas` 中创建一个接口类`EchoService`，添加`@FeignClient`注解，并配置对应的 HTTP URL 地址及 HTTP 方法。

```
package com.aliware.edas;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@FeignClient(name = "service-provider")
public interface EchoService {
    @RequestMapping(value = "/echo/{str}", method = RequestMet
hod.GET)
    String echo(@PathVariable("str") String str);
}
```

      B. 在 Package `com.aliware.edas` 中创建启动类`ConsumerApplication`并添加相关配置。

      - 使用 `@EnableDiscoveryClient` 注解启用服务注册与发现。
      - 使用 `@EnableFeignClients` 注解激活 FeignClient。
      - 添加 `@LoadBalanced` 注解将 RestTemplate 与服务发现集成。

```
package com.aliware.edas;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBoot
Application;
import org.springframework.cloud.client.discovery.EnableDisc
overyClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanc
ed;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class ConsumerApplication {

    @LoadBalanced
    @Bean
```

```
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class, args);
    }
}
```

4. **创建 Controller**。

   在 Package `com.aliware.edas` **中创建类** `TestController` **以演示和验证服务发现功能**。

```
package com.aliware.edas;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class TestController {

    @Autowired
    private RestTemplate restTemplate;
    @Autowired
    private EchoService echoService;

    @RequestMapping(value = "/echo-rest/{str}", method = RequestMet
hod.GET)
    public String rest(@PathVariable String str) {
        return restTemplate.getForObject("http://service-provider/
echo/" + str,
                String.class);
    }

    @RequestMapping(value = "/echo-feign/{str}", method = RequestMet
hod.GET)
    public String feign(@PathVariable String str) {
        return echoService.echo(str);
    }

}
```

5. **修改配置**。

   在`src\main\resources`**路径下创建文件**`application.properties`，**在**`application.properties`**中添加如下配置，指定 Nacos Server 的IP地址**。

```
spring.application.name=service-consumer
server.port=18082
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

其中127.0.0.1:8848为 Nacos Server 的地址。如果您的 Nacos Server 部署在另外一台机器，则需要修改成对应的地址。如果有其它需求，可以参考配置项参考在application.properties文件中增加配置。

6. 查询应用服务。
   a. 执行nacos-service-consumer 中 ConsumerApplication 的 main 函数，启动应用。
   b. 登录本地启动的 Nacos Server 控制台 http://127.0.0.1:8848/nacos（本地 Nacos 控制台的默认用户名和密码同为 nacos），在左侧导航栏中选择服务管理 > 服务列表，可以看到服务列表中已经包含了service-consumer，且在详情中可以查询该服务的详情。

步骤四：查看调用结果

在本地测试消费者对提供者的服务调用结果。启动服务，查看调用结果。

- Linux/Unix/Mac 系统：执行curl http://127.0.0.1:18082/echo-rest/rest-rest 和curl http://127.0.0.1:18082/echo-feign/feign-rest。
- Windows系统：在浏览器中输入http://127.0.0.1:18082/echo-rest/rest-rest和 http://127.0.0.1:18082/echo-feign/feign-rest。

步骤五：将应用部署到 SAE

在本地完成应用的开发和测试后，便可将应用打包并部署到 SAE。部署应用的详细步骤请参见部署应用概述。

> 注意:
> - SAE 暂不支持创建空应用，故第一次部署需在控制台完成。
> - 如果使用 JAR 包部署，在应用部署配置时选择应用运行环境为标准 Java 应用运行环境。
> - 如果使用 WAR 包部署，在应用部署配置时应用运行环境为apache-tomcat-XXX。

当您将应用部署到 SAE 时，SAE 服务注册中心会以更高优先级去设置 Nacos Server 服务端地址和服务端口，以及 namespace、access-key、secret-key、context-path 信息。您无需进行任何额外的配置，原有的配置内容可以选择保留或删除。

结果验证

1. 为步骤五：将应用部署到 SAE中部署的消费者服务应用绑定公网 SLB，在浏览器输入配置好的公网访问地址，进入应用首页。

2. 在应用首页发起调用请求，然后登录 SAE 控制台，进入消费者应用详情页面，在左侧导航栏选择应用监控 > 应用总览，查看服务调用数据总览。

   当能够监测到调用数据则说明服务调用成功。

配置项参考

配置项	Key	默认值	说明
服务端地址	spring.cloud.nacos.discovery.server-addr	无	Nacos Server 启动监听的 IP 地址和端口。
服务名	spring.cloud.nacos.discovery.service	${spring.application.name}	给当前的服务命名。
网卡名	spring.cloud.nacos.discovery.network-interface	无	当 IP 未配置时，注册的 IP 为此网卡所对应的 IP 地址。如果此项也未配置，则默认取第一块网卡的地址。
注册的 IP 地址	spring.cloud.nacos.discovery.ip	无	优先级最高。
注册的端口	spring.cloud.nacos.discovery.port	-1	默认情况下不用配置，系统会自动探测。
命名空间	spring.cloud.nacos.discovery.namespace	无	常用场景之一是不同环境的注册的区分隔离，例如开发测试环境和生产环境的资源（如配置、服务）隔离等。
Metadata	spring.cloud.nacos.discovery.metadata	无	使用 Map 格式配置，用户可以根据自己的需要自定义一些和服务相关的元数据信息。
集群	spring.cloud.nacos.discovery.cluster-name	DEFAULT	配置成 Nacos 集群名称。
接入点	spring.cloud.nacos.discovery.enpoint	UTF-8	地域的某个服务的入口域名，通过此域名可以动态地拿到服务端地址，此配置在部署到 SAE 时无需填写。
是否集成 Ribbon	ribbon.nacos.enabled	true	一般不需要修改。

更多信息

- 更多关于 Spring Cloud Alibaba Nacos Discovery 的信息请参见：Spring Cloud Alibaba Nacos Discovery。
- 如果您在本地开发了依赖 Eureka、Consul、ZooKeeper 等组件实现的服务注册与发现的 Spring Cloud 应用，将该应用修改依赖配置并部署至 SAE 的相关操作请参见：将 Spring Cloud 应用托管到 SAE。

1.3 实现负载均衡

Spring Cloud 的负载均衡是通过 Ribbon 组件完成的。 Ribbon 主要提供客户侧的软件负载均衡算法。Spring Cloud 中的 RestTemplate 和 Feign 客户端底层的负载均衡都是通过 Ribbon 实现的，本章介绍如何在您的应用中实现 RestTemplate 和 Feign 的负载均衡用法。

背景信息

Spring Cloud AliCloud Ans 集成了 Ribbon 的功能，AnsServerList 实现了 Ribbon 提供的 com.netflix.loadbalancer.ServerList 接口。

这个接口是通用的，其它类似的服务发现组件例如 Nacos、Eureka、Consul、ZooKeeper 也都实现了对应的 ServerList 接口，例如 NacosServerList、 DomainExtractingServerList、ConsulServerList、ZookeeperServerList 等。

实现该接口相当于接入了 Spring Cloud 负载均衡规范，这个规范是共用的。这也意味着，从 Eureka、Consul、ZooKeeper 等服务发现方案切换到 Spring Cloud Alibaba 方案，在负载均衡这个层面，无需修改任何代码，RestTemplate、FeignClient，包括已过时的 AsyncRestTemplate，都是如此。

操作步骤

RestTemplate 和 Feign 的实现方式有所不同。

- RestTemplate

  RestTemplate 是 Spring 提供的用于访问 REST 服务的客户端，提供了多种便捷访问远程 HTTP 服务的方法，能够大大提高客户端的编写效率。

   说明:

本文仅提供实现 RestTemplate 和 Feign 的负载均衡代主要代码的造方法，如果您想了解完整的 Spring Cloud 程序，可下载 service-provider和 service-consumer。

您需要在您的应用中按照下面的示例修改代码，以便使用 RestTemplate 的负载均衡。

```
public class MyApp {
    // 注入刚刚使用 @LoadBalanced 注解修饰构造的 RestTemplate
    // 该注解相当于给 RestTemplate 加上了一个拦截器：LoadBalancerIntercep
tor
    // LoadBalancerInterceptor 内部会使用 LoadBalancerClient 接口的实现
类 RibbonLoadBalancerClient 完成负载均衡
    @Autowired
    private RestTemplate restTemplate;
    @LoadBalanced // 使用 @LoadBalanced 注解修改构造的 RestTemplate，使
其拥有一个负载均衡功能
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    // 使用 RestTemplate 调用服务，内部会使用负载均衡调用服务
    public void doSomething() {
        Foo foo = restTemplate.getForObject("http://service-provider
/query", Foo.class);
        doWithFoo(foo);
    }

    ...
}
```

- Feign

  Feign 是 Java 实现的 HTTP 客户端，用于简化 RESTful 调用。

  1. 要想在 Feign 上使用负载均衡，需要添加 Ribbon 的依赖。

     ```
     <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
      <version>{version}</version>
     </dependency>
     ```

  2. 配合 @EnableFeignClients 和 @FeignClient 完成负载均衡请求。

     a. 使用 @EnableFeignClients 开启 Feign 功能。

     ```
     @SpringBootApplication
     @EnableFeignClients // 开启 Feign 功能
     public class MyApplication {
     ...
     }
     ```

     b. 使用 @FeignClient 构造 FeignClient。

     ```
     @FeignClient(name = "service-provider")
     public interface EchoService {
     ```

```
 @RequestMapping(value = "/echo/{str}", method = RequestMethod.
GET)
 String echo(@PathVariable("str") String str);
}
```

c. 注入 EchoService 并完成 echo 方法的调用。

调用 echo 方法相当于发起了一个 HTTP 请求。

```
public class MyService {
   @Autowired // 注入刚刚使用 @FeignClient 注解修饰构造的 EchoServic
e
private EchoService echoService;

public void doSomething() {
        // 相当于发起了一个 http://service-provider/echo/test 请求
        echoService.echo("test");
   }
...
}
```

结果验证

`service-consumer`和多个`serveice-provider`启动后，访问`service-consumer`提供的URL 确认是否实现了负载均衡。

- RestTemplate

  多次访问`/echo-rest/rest-test`查看是否转发到不同的实例。

- Feign

  多次访问`/echo-feign/feign-test`查看是否转发到不同的实例。

1.4 实现配置管理

SAE 将应用配置管理 ACM（Nacos 的正式商用组件）集成到控制台中，提供对应用的配置管理功能。本文以某Spring Cloud 应用的配置为例，介绍如何在本地接入 Nacos，并部署到 SAE 中，使用 ACM 实现配置管理功能。

背景信息

SAE 配置管理中心是开源 Nacos Server 的正式商用组件，以开源版本`Spring Cloud Alibaba Nacos Config`开发的应用可以直接使用 SAE 提供的商业版配置管理中心。

商业版的 SAE 配置管理中心，与 Nacos、Eureka、Zookeeper 和 Spring Cloud Config 相比，具有以下优势：

- 共享组件：节省了你部署、运维 Nacos、Consul、ZooKeeper 或 Spring Cloud Config Server 的成本。

- 实时推送：与 Spring Cloud Config 相比，商业版的 SAE 配置管理中心支持修改过的配置自动推送到监听的客户端。
- 配置推送跟踪：可查询所有客户端配置推送状态和轨迹。

实现SAE 配置管理所需的工程师代码能力门栏低、操作简单。

- 如果您完全不了解 Spring Cloud，只有简单的 Spring 和 Maven 基础。详细阅读本文后，您将了解如何使用 Spring Cloud Alibaba Nacos Config 实现 Spring Cloud 应用的配置管理。
- 如果您熟悉 Spring Cloud 中配置管理组件（如 Consul Config 和 Spring Cloud Config），但尚未使用过 Spring Cloud Alibaba 的配置管理组件 Nacos Config，那么您只需要将这些配置组件的依赖和配置替换成 Spring Cloud Alibaba Nacos Config，并修改相应的配置项即可，无需修改任何代码。

  Spring Cloud Alibaba Nacos Config 同样实现了 Spring Cloud Config 的标准接口与规范，和您之前接入配置管理的方式基本一致。
- 如果您已经熟悉如何通过开源版本的 Spring Cloud Alibaba Nacos Config 实现 Spring Cloud 应用的配置管理，那么您可以将应用直接部署到 SAE（详情请参见部署到 SAE），即可使用到 SAE 提供的商业版配置管理的能力。

本地开发

本文仅介绍实现SAE配置管理功能主要代码的改造方法，如果您想了解完整的 Spring Cloud 程序，可下载 nacos-config-example]。

说明:
Spring Cloud Alibaba Nacos Config 完成了 Nacos 与 Spring Cloud 框架的整合，支持 Spring Cloud 的配置注入规范。

1. 准备工作。

   在开始代码改造前，请确保您已经完成如下工作：

项目	操作说明
下载 Maven 并设置环境变量。	下载 Maven 并设置环境变量。
下载最新版本的 Nacos Server。	下载最新版本的 Nacos Server。

项目	操作说明
启动 Nacos Server。	a. 解压下载的 Nacos Server 压缩包。 b. 进入`nacos/bin`目录，启动 Nacos Server。 Linux/Unix/Mac 系统：执行命令`sh startup.sh -m standalone`。 Windows 系统：双击执行`startup.cmd`文件。
在本地 Nacos Server 控制台新建配置。	a. 登录本地 Nacos Server 控制台（用户名和密码默认同为 nacos） b. 在左侧导航栏中单击配置列表，在配置列表页面右上角单击新建配置图标 ＋ 。 c. 在新建配置页面填入以下信息，然后单击发布。 Data ID: *nacos-config-example.properties* Group: *DEFAULT_GROUP* 配置内容： *test.name=nacos-config-test*

2. 使用 Nacos Config 实现配置管理。

   a. 创建名为`nacos-config-example`的 Maven 工程。

   b. 在`pom.xml`文件中添加依赖。

   以 Spring Boot 2.1.4.RELEASE 和 Spring Cloud Greenwich.SR1 为例，依赖如下：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
```

```
        <artifactId>spring-cloud-starter-alibaba-nacos-config</
artifactId>
        <version>0.9.0.RELEASE</version>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Greenwich.SR1</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

示例中使用的版本为 Spring Cloud Greenwich，对应 Spring Cloud Alibaba 版本为 0.9.0.RELEASE。

- **如果使用 Spring Cloud Finchley 版本，对应 Spring Cloud Alibaba 版本为 0.2.2.RELEASE。**
- **如果使用 Spring Cloud Edgware 版本，对应 Spring Cloud Alibaba 版本为 0.1.2.RELEASE。**

说明:

Spring Cloud Edgware 版本的生命周期即将在 2019 年 8 月结束，不推荐使用这个版本开发应用。

c. **在**`src\main\java`**下创建 Package**`com.aliware.sae`。

d. **在 Package**`com.aliware.sae`**中创建**`nacos-config-example`**的启动类** `NacosConfigExampleApplication`。

```
    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBoot
Application;

    @SpringBootApplication
    public class NacosConfigExampleApplication {
        public static void main(String[] args) {
            SpringApplication.run(NacosConfigExampleApplication.
class, args);
        }
```

```
    }
```

e. 在 Package`com.aliware.sae`中创建 Controller`EchoController`，自动注入一个属性`userName`，且通过`@Value`注解指定中从配置中 Key 为`test.name`去取值。

```
    import org.springframework.beans.factory.annotation.Value;
    import org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RestController;

    @RestController
    @RefreshScope
    public class EchoController {

        @Value("${test.name}")
        private String userName;

        @RequestMapping(value = "/")
        public String echo() {
            return userName;
        }
    }
```

f. 在`src\main\resources`路径下创建文件`bootstrap.properties`，在`bootstrap.properties`中添加如下配置，指定 Nacos Server 的IP地址。

其中 127.0.0.1:8848 为 Nacos Server 的地址，如果您的 Nacos Server 部署在另外一台机器，则需要修改成对应的 IP 和 端口。如果有其它需求，可以参考配置项参考在`bootstrap.properties`文件中增加配置。

```
spring.application.name=nacos-config-example
server.port=18081
spring.cloud.nacos.config.server-addr=127.0.0.1:8848
```

g. 执行`NacosConfigExampleApplication`中的 main 函数，启动应用。

3. 结果验证。

在浏览器访问 http://127.0.0.1:18081 ，可以看到返回值为 `nacos-config-test` ，该值即为在本地 Nacos Server 中新建配置中的配置内容，即`test.name`的值。

部署到 SAE

当在本地完成应用的开发和测试后，便可将应用程序打包并部署到 SAE。部署应用的详细步骤请参见#unique_10。

SAE 配置管理中心提供了正式商用版本 Nacos Server。当您将应用部署到 SAE 的时候，SAE 会通过优先级更高的方式去设置 Nacos Server 服务端地址和服务端口，以及 namespace、access-key、secret-key、context-path 信息。您无需进行任何额外的配置，原有的配置内容可以选择保留或删除。

- 在部署应用前，需要在 SAE 控制台的配置管理中新建和本地 Nacos Server 中相同的配置，具体操作步骤如下：

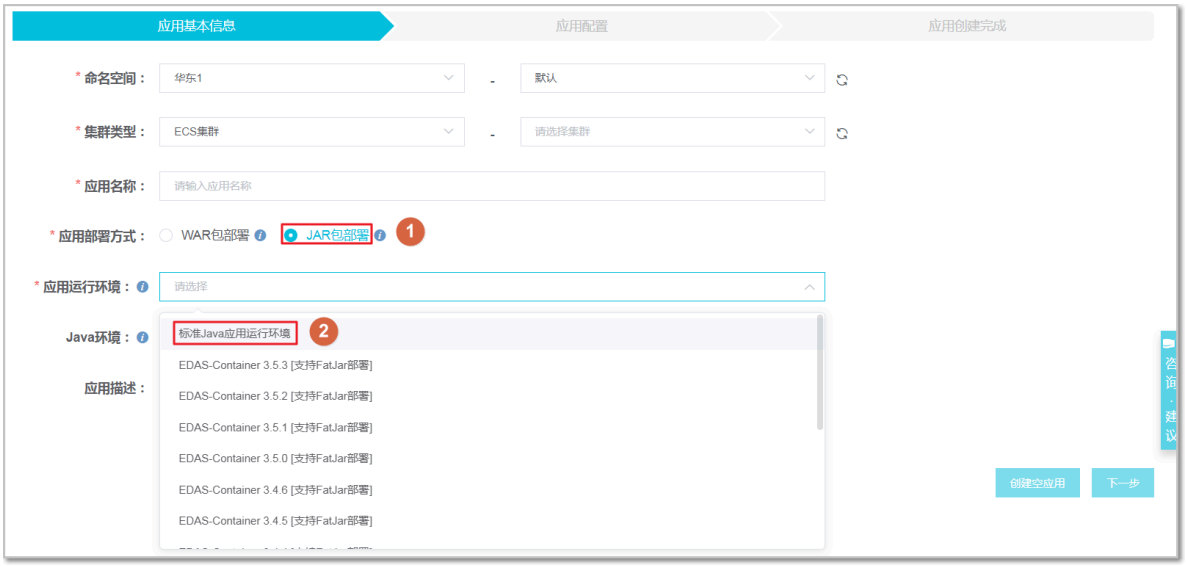
  1. 登录 SAE 控制台。
  2. 在左侧导航栏中选择配置管理。
  3. 在配置管理页面选择地域和命名空间，单击页面右侧 ＋。
  4. 在新建配置页面设置 Data ID、Group 和 配置内容，然后单击发布。

     Data ID: *nacos-config-example.properties*

     Group: *DEFAULT_GROUP*

     配置内容: *test.name=nacos-config-test*

- 初次通过SAE部署应用建议在控制台进行部署。如果使用 JAR 包部署，在创建应用时应用运行环境 须选 标准 Java 应用运行环境。



结果验证

1. 部署完成后，可以通过查看日志确认应用是否启动成功。
2. 执行命令`curl http://<应用实例 IP>:<服务端口>`，如`curl http://192.168.0.34:8080` 查看是否返回配置内容 *nacos-config-test*。
3. 在 SAE 控制台将原有配置内容修改为 *nacos-config-test2*，再执行命令`curl http://<应用实例 IP>:<服务端口>`，如`curl http://192.168.0.34:8080`，查看是否返回变更后的配置内容 *nacos-config-test2*。

配置项参考

配置项	key	默认值	说明
服务端地址	spring.cloud.nacos.config.server-addr	无	无
DataId 前缀	spring.cloud.nacos.config.prefix	${spring.application.name}	dataId 的前缀
Group	spring.cloud.nacos.config.group	DEFAULT_GROUP	
dataID 后缀及内容文件格式	spring.cloud.nacos.config.file-extension	properties	dataId 的后缀，同时也是配置内容的文件格式，默认是 properties，支持 yaml 和 yml。
配置内容的编码方式	spring.cloud.nacos.config.encode	UTF-8	配置的编码
获取配置的超时时间	spring.cloud.nacos.config.timeout	3000	单位为 ms
配置的命名空间	spring.cloud.nacos.config.namespace		常用场景之一是不同环境的配置的区分隔离，例如开发测试环境和生产环境的资源隔离等。
相对路径	spring.cloud.nacos.config.context-path		服务端 API 的相对路径
接入点	spring.cloud.nacos.config.endpoint	UTF-8	地域的某个服务的入口域名，通过此域名可以动态地拿到服务端地址。
是否开启监听和自动刷新	spring.cloud.nacos.config.refresh.enabled	true	默认为 true，不需要修改。

更多配置项，请参考开源版本的 Spring Cloud Alibaba Nacos Config 文档。

1.5 搭建服务网关

本文介绍如何基于 Spring Cloud Gateway 和 Spring Cloud Netflix Zuul 使用 Nacos 搭建应用的服务网关。

背景信息

SAE 服务注册中心提供了开源 Nacos Server 的正式商用版本，使用开源版本 `Spring Cloud Alibaba Nacos Discovery`开发的应用可以直接使用 SAE 提供的商业版服务注册中心。

商业版的 SAE 服务注册中心，与开源版本的 Nacos、Eureka 和 Consul 相比，还具有以下优势：

- 共享组件，节省了你部署运维 Nacos、Eureka 或 Consul 的成本。
- 在服务注册和发现的调用中都进行了链路加密，保护您的服务，无需再担心服务被未授权的应用发现。
- SAE 服务注册中心 与 SAE 其他组件紧密结合，为您提供一整套的微服务解决方案，包括 环境隔离、平滑上下线、灰度发布等。

本文仅提供服务网关搭建过程中主要代码的改造方法，如果您想了解完整的 Spring Cloud 程序，可下载 spring-cloud-gateway-nacos、spring-cloud-zuul-nacos 和 nacos-service-provider。

准备工作

- 已下载 Maven 并设置环境变量。
- 已下载最新版本的 Nacos Server并启动。

> **说明:**
>
> 启动 Nacos Server
>
> 1. 解压已下载的 Nacos Server 压缩包。
> 2. 进入`nacos/bin`目录，启动 Nacos Server。
>
> - Linux/Unix/Mac 系统：执行命令`sh startup.sh -m standalone`。
> - Windows 系统：双击执行`startup.cmd`文件。

基于 Spring Cloud Gateway 搭建服务网关。

- 创建服务网管关

1. 创建名为`spring-cloud-gateway-nacos`的 Maven 工程。
2. 在`pom.xml`文件中添加 Spring Boot 和 Spring Cloud 的依赖。

   本文以 Spring Boot 2.1.4.RELEASE 和 Spring Cloud Greenwich.SR1 版本为例。

```
 <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>2.1.4.RELEASE</version>
     <relativePath/>
 </parent>

 <dependencies>
     <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-starter-gateway</artifactId>
     </dependency>
     <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-starter-alibaba-nacos-discovery
</artifactId>
         <version>0.9.0.RELEASE</version>
     </dependency>
 </dependencies>

 <dependencyManagement>
     <dependencies>
         <dependency>
             <groupId>org.springframework.cloud</groupId>
             <artifactId>spring-cloud-dependencies</artifactId>
             <version>Greenwich.SR1</version>
             <type>pom</type>
             <scope>import</scope>
         </dependency>
     </dependencies>
 </dependencyManagement>

 <build>
     <plugins>
         <plugin>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
     </plugins>
 </build>
```

3. 创建服务网关启动类`GatewayApplication`。

```
    @SpringBootApplication
    @EnableDiscoveryClient
    public class GatewayApplication {
        public static void main(String[] args) {
            SpringApplication.run(GatewayApplication.class, args);
        }
    }
```

4. 在application.yaml中添加如下配置，将注册中心指定为 Nacos Server 的地址。

   其中127.0.0.1:8848为 Nacos Server 的IP地址。如果您的 Nacos Server 部署在另外一台机器，则需要修改成对应的地址。

   其中 routes 配置了 Gateway 的路由转发策略，本示例中此处将所有前缀为/provider1/的请求，全部由路由器分配到服务名为service-provider的后端服务中。

```
    server:
  port: 15012

    spring:
  application:
        name: spring-cloud-gateway-nacos
  cloud:
        gateway: # config the routes for gateway
          routes:
          - id: service-provider          # 将 /provider1/ 开头的请
求转发到 provider1
            uri: lb://service-provider
            predicates:
            - Path=/provider1/**
            filters:
            - StripPrefix=1               # 表明前缀 /provider1 需要
截取掉
        nacos:
          discovery:
            server-addr: 127.0.0.1:8848
```

5. 执行启动类GatewayApplication中的 main 函数，启动 Gateway。
6. 登录本地已启动的本地 Nacos Server 控制台 *http://127.0.0.1:8848/nacos* （本地 Nacos 控制台的默认用户名和密码同为 nacos）。在左侧导航栏中选择服务管理 > 服务列表，可以看到服务列表中已经包含了 *spring-cloud-gateway-nacos*，且在详情页中可以查询该服务的详细信息。表示服务网关已经启动并注册成功，成功后需要通过创建下游服务验证网关的请求转发功能。

- 创建服务提供者

  创建服务提供者应用，如何创建具体请参考将 Spring Cloud 应用托管到 SAE。

  服务提供者示例：

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProviderApplication {

    public static void main(String[] args) {

        SpringApplication.run(ProviderApplication, args);
    }

    @RestController
```

```
    public class EchoController {
        @RequestMapping(value = "/echo/{string}", method =
RequestMethod.GET)
        public String echo(@PathVariable String string) {
            return string;
        }
    }
}
```

- 结果验证
  - 本地验证

    本地启动已经开发好的服务网关和服务提供者，访问 Spring Cloud Gateway 将请求转发给后端服务，可以看到调用成功后返回的结果。

```
➜  spring-cloud-gateway-nacos curl http://127.0.0.1:18012/provider1/echo/123456
123456%
```

  - 在 SAE 中验证

    请参考应用部署概述将您的应用部署到 SAE，并进行验证。

    SAE 服务注册中心提供了正式商用版本 Nacos Server。当您将应用部署到 SAE 的时候，SAE 会通过优先级更高的方式去设置 Nacos Server 服务端地址和服务端口，以及 namespace、access-key、secret-key、context-path 信息。您无需进行任何额外的配置，原有的配置内容可以选择保留或删除。

基于 Zuul 搭建服务网关

介绍如何基于 Zuul 使用 Nacos 作为服务注册中心搭建应用的服务网关。

- 创建服务网关

1. 创建名为`spring-cloud-zuul-nacos`的Maven 工程。
2. 在`pom.xml`文件中添加 Spring Boot、Spring Cloud 和 Spring Cloud Alibaba 的依赖。

   Spring Boot： 2.1.4.RELEASE、Spring Cloud： Greenwich.SR1 和 Spring Cloud：Alibaba 0.9.0 。

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <dependency>
```

```
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-discovery
</artifactId>
            <version>0.9.0.RELEASE</version>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Greenwich.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

3. **创建服务网关启动类**`ZuulApplication`。

```
    @SpringBootApplication
    @EnableZuulProxy
    @EnableDiscoveryClient
    public class ZuulApplication {
        public static void main(String[] args) {
            SpringApplication.run(ZuulApplication.class, args);
        }
    }
```

4. **在**`application.properties`**中添加如下配置，将注册中心指定为 Nacos Server 的IP地址**。

```
spring.application.name=spring-cloud-zuul-nacos
server.port=18022

spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

zuul.routes.opensource-provider1.path=/provider1/**
zuul.routes.opensource-provider1.serviceId=service-provider
```

`127.0.0.1:8848`为 Nacos Server 的IP地址。如果您的 Nacos Server 部署在另外一台机器，则需要修改成对应的IP地址。

代码中 routes 配置了 Zuul 的路由转发策略，本示例中此处将所有前缀为`/provider1/`的请求，全部通过路由分配到服务名为`service-provider`的后端服务中。

5. 执行 spring-cloud-zuul-nacos 中的 main 函数`ZuulApplication`，启动服务。
6. 登录本地已启动的 Nacos Server 控制台 *http://127.0.0.1:8848/nacos* （本地 Nacos 控制台的默认用户名和密码同为 nacos），在左侧导航栏中选择服务管理 > 服务列表，可以看到服务列表中已经包含了 s*pring-cloud-zuul-nacos*，且在详情页中可以查询该服务的详细信息。表示服务网关已经启动并注册成功，成功后需要通过创建下游服务验证网关的请求转发功能。

- 创建服务提供者

如何快速创建服务提供者，具体操作请参见将 Spring Cloud 应用托管到 SAE。

服务提供者启动类示例：

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProviderApplication {

    public static void main(String[] args) {

        SpringApplication.run(ProviderApplication, args);
    }

    @RestController
    public class EchoController {
        @RequestMapping(value = "/echo/{string}", method =
RequestMethod.GET)
        public String echo(@PathVariable String string) {
            return string;
        }
    }
}
```

- 结果验证

1. 本地验证。

   本地启动已开发好的服务网关 Zuul 和服务提供者，访问 Spring Cloud Netflix Zuul 将请求转发给后端服务，可以看到调用成功后返回的结果。

```
➜  spring-cloud-gateway-nacos curl http://127.0.0.1:18022/provider1/echo/123456
123456%
➜  spring-cloud-gateway-nacos
```

2. 在 SAE 中验证。

   您可以参考应用部署概述将您的应用部署到 SAE，并进行验证。

   SAE 服务注册中心提供了正式商用版本 Nacos Server。当您将应用部署到 SAE 的时候，SAE 会通过优先级更高的方式去设置 Nacos Server 服务端地址和服务端口，以及 namespace、access-key、secret-key、context-path 信息。您无需进行任何额外的配置，原有的配置内容可以选择保留或删除。

FAQ

1. 使用其他版本

   示例中使用的 Spring Cloud 版本为 Greenwich，对应的 Spring Cloud Alibaba 版本为 0.9.0.RELEASE。Spring Cloud Finchley 对应的 Spring Cloud Alibaba 版本为 0.2.2.RELEASE，Spring Cloud Edgware 对应的 Spring Cloud Alibaba 版本为 0.1.2.RELEASE。

   > 说明:
   > Spring Cloud Edgware 版本的生命周期即将在 2019 年 8 月结束，不推荐使用这个版本开发应用。

2. 从 ANS 迁移

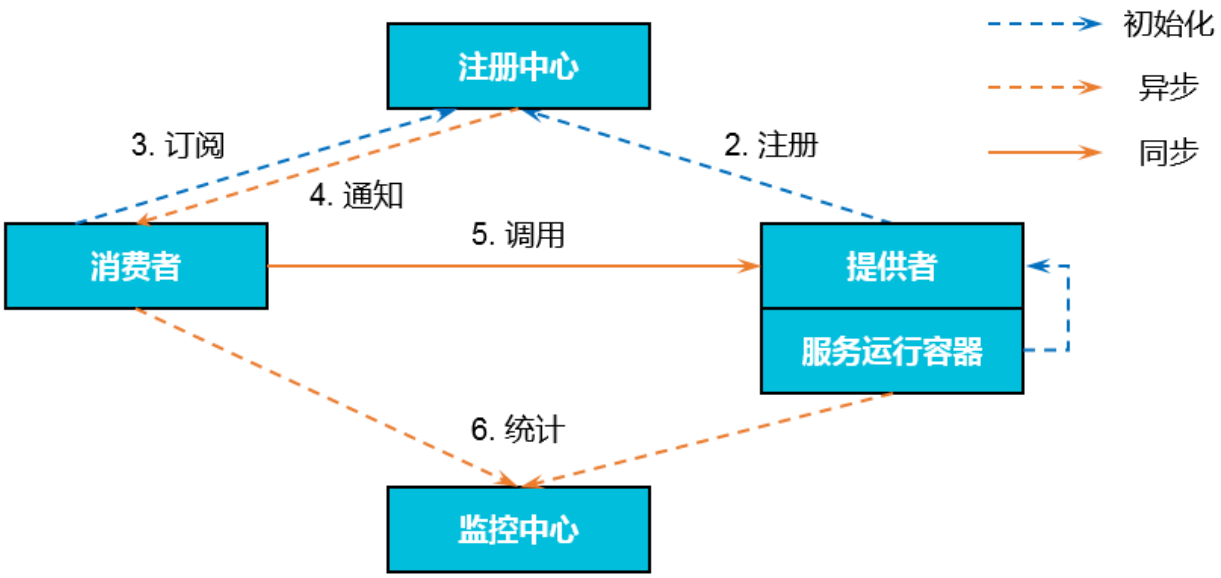
   SAE 注册中心在服务端对 ANS 和 Nacos 的数据结构做了兼容，在同一个命名空间下，且 Nacos 未设置 group 时，Nacos 和 ANS 客户端可以互相发现对方注册的服务。

2 Dubbo 开发

2.1 Dubbo 开发概述

SAE 支持原生 Dubbo 微服务框架，您在这个框架下开发的微服务只需添加依赖和修改配置，即可获取 SAE 企业级的微服务应用托管、微服务治理、监控报警和应用诊断等能力，实现零代码量应用迁移。

Dubbo 的架构如下图所示。



1. 服务运行容器负责启动，加载，运行提供者服务。
2. 提供者在启动时，需要向注册中心进行注册。
3. 消费者在启动时，需要向注册中心订阅所需的服务。
4. 注册中心返回提供者地址列表给消费者。如果有变更，注册中心将基于长连接推送变更数据给消费者。
5. 消费者从提供者地址列表中，基于软负载均衡算法，选择某个提供者进行调用。如果调用失败，则重新调用其他提供者。
6. 消费者和提供者在内存中存储累计调用次数和调用时间，定时（每分钟）发送统计数据至监控中心。

2.2 初次使用 Dubbo 构建微服务应用

当您初次使用原生 Dubbo 构建微服务应用时，您需在本地完成添加依赖和配置管理等操作，然后将应用部署到 SAE，然后实现微服务应用的服务注册与发现，并查看服务调用关系。

背景信息

如果您完全不了解 Dubbo，详细阅读本文后，您将了解如何通过 `sae-dubbo-extension` 实现 Dubbo 应用的服务注册与发现，以及实现消费者对提供者的服务调用。

SAE 目前支持的 Dubbo 版本为 2.5.3 到 2.7.0：

- 如果您使用的 Dubbo 版本为 *2.5.3 ~ 2.6.x*，对应的 `sae-dubbo-extension` 的版本为 *1.0.6*。
- 如果您使用的 Dubbo 版本为 *2.7.0*，对应的 `sae-dubbo-extension` 的版本改成 *2.0.2*。

SAE 服务注册中心实现了 Dubbo 所提供的 SPI 标准的注册中心扩展，能够完整地支持 Dubbo 服务注册、路由规则、配置规则功能。

SAE 服务注册中心能够完全代替 ZooKeeper 和 Redis，作为您 Dubbo 服务的注册中心。同时，与 ZooKeeper 和 Redis 相比，还具有以下优势：

- SAE 服务注册中心为共享组件，节省了您运维、部署 ZooKeeper 等组件的机器成本。
- SAE 服务注册中心在通信过程中增加了鉴权加密功能，为您的服务注册链路进行了安全加固。
- SAE 服务注册中心与 SAE 其他组件紧密结合，为您提供一整套的微服务解决方案。

准备工作

- 下载、启动并配置轻量级配置中心。

  为了便于本地开发，SAE 提供了包含 SAE 服务注册中心基本功能的轻量级配置中心。基于轻量级配置中心开发的应用无需修改任何代码和配置就可以部署到 SAE 中。

  请您参考配置轻量级配置中心进行下载、启动及配置。推荐使用最新版本。
- 下载 Maven 并设置环境变量（如已安装可跳过）。

步骤一：获取 Demo

sae-dubbo-demo 下载

步骤二：创建服务提供者

在本地创建一个提供者应用工程，添加依赖，配置服务注册与发现，并将注册中心指定为 SAE 轻量级配置中心。

1. 使用 IDE（如 IDEA 和 Eclipse）创建一个 Maven 工程。

2. 在 Maven 工程中的`pom.xml`文件中添加 dubbo 和 sae-dubbo-extension 依赖，版本分别为 *2.6.2* 和 *1.0.6*。

```
<dependencies>
     <dependency>
         <groupId>com.alibaba</groupId>
         <artifactId>dubbo</artifactId>
         <version>2.6.2</version>
     </dependency>

     <dependency>
         <groupId>com.alibaba.sae</groupId>
         <artifactId>sae-dubbo-extension</artifactId>
         <version>1.0.6</version>
     </dependency>
</dependencies>
```

3. 开发 Dubbo 服务提供者。

   Dubbo 中服务是以接口的形式提供。

   a. 在`src/main/java`路径下创建名为`com.alibaba.sae` package 。

   b. 在`com.alibaba.sae`下创建名为 `IHelloService`接口（interface），包含 `SayHello` 方法。

```
package com.alibaba.sae;

public interface IHelloService {
    String sayHello(String str);
}
```

   c. 在`com.alibaba.sae`下创建名为`IHelloServiceImpl`类，用来实现此接口。

```
package com.alibaba.sae;

public class IHelloServiceImpl implements IHelloService {
    public String sayHello(String str) {
        return "hello " + str;
    }
}
```

4. 配置 Dubbo 服务。

   a. 在 `src/main/resources`路径下创建 `provider.xml`文件并打开。

   b. 在`provider.xml`中，添加 Spring 相关的 XML NameSpace（xmlns） 和 XML Schema Instance（xmlns:xsi），以及 Dubbo 相关的 NameSpace（xmlns:dubbo） 和 Scheme Instance（xsi:schemaLocation）。

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
```

```
     xsi:schemaLocation="http://www.springframework.org/schema/
beans
     http://www.springframework.org/schema/beans/spring-beans-4.0.
xsd
     http://code.alibabatech.com/schema/dubbo
     http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
```

c. 在 `provider.xml` 中将接口和实现类展现为 Dubbo 服务。

```
  <dubbo:application name="demo-provider"/>

  <dubbo:protocol name="dubbo" port="28082"/>

  <dubbo:service interface="com.alibaba.sae.IHelloService" ref="
helloService"/>

  <bean id="helloService" class="com.alibaba.sae.IHelloServiceImpl
"/>
```

d. 在 `provider.xml` 中将注册中心指定为 SAE 轻量级配置中心。

其中 *127.0.0.1* 为轻量级配置中心的地址，如果您的轻量级配置中心部署在另外一台机器，则需要修改成对应的 IP 地址。由于轻量级配置中心不支持修改端口，所以端口必须使用 8080。

```
  <dubbo:registry id="sae" address="sae://127.0.0.1:8080"/>
```

5. 启动服务。

a. 在 `com.alibaba.sae`中创建类 Provider，并按下面的代码在 Provider 的 main 函数中加载 Spring Context，将配置好的 Dubbo 服务展现。

```
package com.alibaba.sae;

import org.springframework.context.support.ClassPathXmlApplicat
ionContext;

public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathX
mlApplicationContext(new String[] {"provider.xml"});
        context.start();
        System.in.read();
    }
}
```

b. 执行 Provider 的 main 函数，启动服务。

6. 登录轻量级配置中心控制台 `http://127.0.0.1:8080`，在左侧导航栏中单击服务列表，查看提供者列表。在列表中可以看到服务提供者已经包含了 `com.alibaba.sae.IHelloService`，且可以查询该服务的服务分组和提供者 IP。

步骤三：创建服务消费者

在本地创建一个消费者应用工程，添加依赖，添加订阅服务的配置。

1. 创建 Maven 项目并引入依赖。

   具体操作请参见步骤二：创建服务提供者的步骤1～步骤2。

2. 开发 Dubbo 服务。

   Dubbo 中服务是以接口的形式提供。

   a. 在`src/main/java`路径下创建名为`com.alibaba.sae`的package 。

   b. 在`com.alibaba.sae`下创建名为`IHelloService`接口（interface），其中包含了`SayHello` 方法。

   说明:

   通常在单独的模块中定义接口供其他程序调用，服务提供者和服务消费者都通过 Maven 依赖来调用此模块。本文档从易操作和便于理解角度出发，服务提供者和服务消费者分别创建了两个完全相同的接口，现场实际开发过程中不推荐创建两个相同接口。

   ```
   package com.alibaba.sae;

   public interface IHelloService {
       String sayHello(String str);
   }
   ```

3. 配置 Dubbo 服务。

   a. 在 `src/main/resources`路径下创建 `consumer.xml`文件并打开。

   b. 在`consumer.xml`中，添加 Spring 相关的 XML NameSpace（xmlns） 和 XML Schema Instance（xmlns:xsi），以及 Dubbo 相关的 NameSpace（xmlns:dubbo）和 Scheme Instance（xsi:schemaLocation）。

   ```
   <beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
      xsi:schemaLocation="http://www.springframework.org/schema/
beans
      http://www.springframework.org/schema/beans/spring-beans-4.0.
xsd
      http://code.alibabatech.com/schema/dubbo
      http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
   ```

   c. 在 `consumer.xml` 中添加如下配置，订阅 Dubbo 服务。

   ```
   <dubbo:application name="demo-consumer"/>

   <dubbo:registry id="sae" address="sae://127.0.0.1:8080"/>
   ```

```
  <dubbo:reference id="helloService" interface="com.alibaba.sae.
IHelloService"/>
```

4. 启动、验证服务。

   a. 在`com.alibaba.sae`下创建类 Consumer，并按下面的代码在 Consumer 的 main 函数中加载 Spring Context，订阅并消费 Dubbo 服务。

```
    package com.alibaba.sae;

    import org.springframework.context.support.ClassPathX
mlApplicationContext;

    import java.util.concurrent.TimeUnit;

    public class Consumer {
        public static void main(String[] args) throws Exception {
            ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext(new String[] {"consumer.xml"});
            context.start();
            while (true) {
                try {
                    TimeUnit.SECONDS.sleep(5);
                    IHelloService demoService = (IHelloService)
context.getBean("helloService");
                    String result = demoService.sayHello("world");
                    System.out.println(result);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

   b. 执行 Consumer 的 main 函数，启动服务。

5. 验证创建结果。

   a. 启动后，控制台不断地输出 `hello world`，表示服务消费成功。

   b. 登录轻量级配置中心控制台 `http://127.0.0.1:8080`，在左侧导航栏中选择服务列表 > 调用者列表，在调用者列表显示 `com.alibaba.sae.IHelloService`，且可以查看该服务的服务分组和调用者 IP。

步骤四：部署到 SAE

1. 分别在 Provider 和 Consumer 的 pom.xml 文件中添加如下配置，然后执行 `mvn clean package` 将本地的程序打成可执行的 JAR 包。

   · Provider

```
<build>
 <plugins>
       <plugin>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                    <configuration>
                        <classifier>spring-boot</classifier>
                        <mainClass>com.alibaba.sae.Provider</
mainClass>
                    </configuration>
                </execution>
            </executions>
        </plugin>
 </plugins>
</build>
```

- Consumer

```
<build>
 <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                    <configuration>
                        <classifier>spring-boot</classifier>
                        <mainClass>com.alibaba.sae.Consumer</
mainClass>
                    </configuration>
                </execution>
            </executions>
        </plugin>
 </plugins>
</build>
```

2. 参考部署微服务应用到 SAE部署应用。

更多信息

您还可以使用 Spring Boot 开发 Dubbo 应用。

2.3 使用 Spring Boot 开发 Dubbo 应用

如果您只有简单的 Java 基础和 Maven 经验，而不熟悉 Dubbo，您可以从零开始开发 Dubbo 服务，并使用 SAE 服务注册中心实现服务注册与发现。

前提条件

- 下载、启动及配置轻量级配置中心。

  为了便于本地开发，SAE 提供了含有 SAE 服务注册中心基本功能的轻量级配置中心。基于轻量级配置中心开发的应用无需修改任何代码和配置就可以部署到云端的 SAE 中。

  请您参考 配置轻量级配置中心 进行下载、启动及配置。推荐使用最新版本。

- 下载 Maven 并设置环境变量（本地已安装的可略过）。

背景信息

SAE 服务注册中心实现了 Dubbo 所提供的 SPI 标准的注册中心扩展，能够完整地支持 Dubbo 服务注册、路由规则、配置规则功能。

SAE 服务注册中心能够完全代替 ZooKeeper 和 Redis，作为您 Dubbo 服务的注册中心。同时，与 ZooKeeper 和 Redis 相比，还具有以下优势：

- SAE 服务注册中心为共享组件，节省了您运维、部署 ZooKeeper 等组件的机器成本。
- SAE 服务注册中心在通信过程中增加了鉴权加密功能，为您的服务注册链路进行了安全加固。
- SAE 服务注册中心与 SAE 其他组件紧密结合，为您提供一整套的微服务解决方案。

全新场景使用 Spring Boot 开发 Dubbo 应用有两种主要的方式：

- 使用 xml 开发
- 使用 Spring Boot 的注解方式开发

使用 xml 方式请参考 将 Dubbo 应用托管到 SAE。文本档介绍如何使用 Spring Boot 的注解方式开发 Dubbo 服务。

Spring Boot 使用极简的配置能够快速搭建基于 Spring 的 Dubbo 服务，相比 xml 的开发方式，开发效率更高。

步骤1：创建服务提供者

1. 创建名为`spring-boot-dubbo-provider` Maven 工程，命。
2. 在`pom.xml`文件中添加所需的依赖。

   这里本文以 Spring Boot 2.0.6.RELEASE 为例。

```
    <dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>2.0.6.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.boot</groupId>
        <artifactId>dubbo-spring-boot-starter</artifactId>
        <version>0.2.0</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba.edas</groupId>
        <artifactId>edas-dubbo-extension</artifactId>
        <version>1.0.6</version>
    </dependency>

</dependencies>
```

如果您需要选择使用 Spring Boot 1.x 的版本，请使用 Spring Boot 1.5.x 版本，对应的 com.alibaba.boot:dubbo-spring-boot-starter 版本为 0.1.0。

说明:

Spring Boot 1.x 版本的生命周期即将在 2019 年 8 月 结束，推荐使用新版本开发您的应用。

3. 开发 Dubbo 服务提供者。

   Dubbo 中服务都是以接口的形式提供的。

   a) 在`src/main/java`路径下创建 package `com.alibaba.sae.boot`。

   b) 在`com.alibaba.sae.boot`下创建名为`IHelloService`接口（interface），其中包含了 `SayHello` 方法。

```
package com.alibaba.sae.boot;
public interface IHelloService {
String sayHello(String str);
}
```

   c) 在`com.alibaba.sae.boot`下创建名为`IHelloServiceImpl`类，实现此接口。

```
package com.alibaba.sae.boot;
import com.alibaba.dubbo.config.annotation.Service;
@Service
public class IHelloServiceImpl implements IHelloService {
public String sayHello(String name) {
  return "Hello, " + name + " (from Dubbo with Spring Boot)";
 }
}
```

说明:

这里的 Service 注解是 Dubbo 提供的一个注解类，类的全名称为：com.alibaba.dubbo.config.annotation.Service 。

4. **配置 Dubbo 服务**

   a) **在** `src/main/resources`**路径下创建**`application.properties`**或**`application.yaml`**文件，并将其打开。**

   b) **在**`application.properties`**或**`application.yaml`**中添加如下配置。**

```
# Base packages to scan Dubbo Components (e.g @Service , @
Reference)
dubbo.scan.basePackages=com.alibaba.sae.boot
dubbo.application.name=dubbo-provider-demo
dubbo.registry.address=edas://127.0.0.1:8080
```

说明:

- **以上三个配置没有默认值，必须要给出具体的配置。**
- `dubbo.scan.basePackages`**的值为开发代码中含有**`com.alibaba.dubbo.config.annotation.Service`**和**`com.alibaba.dubbo.config.annotation.Reference`**注解所在的包。多个包之间用逗号隔开。**
- `dubbo.registry.address`**的值前缀必须以** *edas://* **开头，后面的 IP 地址和端口是轻量版配置中心。**

5. **开发并启动 Spring Boot 入口类**`DubboProvider`。

```
    package com.alibaba.sae.boot;

    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBoot
Application;

    @SpringBootApplication
    public class DubboProvider {

        public static void main(String[] args) {

            SpringApplication.run(DubboProvider.class, args);
        }

    }
```

6. 登录轻量版配置中心控制台，在左侧导航栏中单击服务列表，查看提供者列表。

   可以看到服务提供者里已经包含了`com.alibaba.sae.boot.IHelloService`，且可以查询该服务的服务分组和提供者 IP。

步骤2：创建服务消费者

1. 创建名为`spring-boot-dubbo-consumer` Maven 工程。
2. 在`pom.xml`文件中添加相关依赖。

   本文以 Spring Boot 2.0.6.RELEASE 为例。

```
    <dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>2.0.6.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba.boot</groupId>
        <artifactId>dubbo-spring-boot-starter</artifactId>
        <version>0.2.0</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba.edas</groupId>
        <artifactId>edas-dubbo-extension</artifactId>
        <version>1.0.6</version>
    </dependency>

</dependencies>
```

   如果您需要选择使用 Spring Boot 1.x 的版本，请使用 Spring Boot 1.5.x 版本，对应的 com.alibaba.boot:dubbo-spring-boot-starter 版本为 0.1.0。

   说明:

   Spring Boot 1.x 版本的生命周期即将在 2019 年 8 月 结束，推荐使用新版本开发您的应用。

3. 开发 Dubbo 消费者

   a) 在`src/main/java`路径下创建 package `com.alibaba.sae.boot`。

   b) 在`com.alibaba.sae.boot`下创建名为`IHelloService`接口（interface），里面包含了`SayHello`方法。

```
package com.alibaba.sae.boot;

public interface IHelloService {
 String sayHello(String str);
}
```

4. 开发 Dubbo 服务调用。

   例如需要在 Controller 中调用一次远程 Dubbo 服务，开发的代码如下所示。

```
package com.alibaba.sae.boot;

import com.alibaba.dubbo.config.annotation.Reference;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
    public class DemoConsumerController {

        @Reference
        private IHelloService demoService;

        @RequestMapping("/sayHello/{name}")
        public String sayHello(@PathVariable String name) {
            return demoService.sayHello(name);
        }
    }
```

说明:

这里的 Reference 注解是 com.alibaba.dubbo.config.annotation.Reference 。

5. 在`application.properties/application.yaml`配置文件中新增以下配置：

```
dubbo.application.name=dubbo-consumer-demo
dubbo.registry.address=edas://127.0.0.1:8080
```

说明:

- 以上两个配置没有默认值，必须要给出具体的配置。
- `dubbo.registry.address`的值前缀必须是以 `edas://` 开头，后面的 IP 地址和端口是轻量版配置中心。

6. 开发并启动 Spring Boot 入口类`DubboConsumer`。

```
package com.alibaba.sae.boot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DubboConsumer {

    public static void main(String[] args) {

        SpringApplication.run(DubboConsumer.class, args);
    }

}
```

7. 登录轻量版配置中心控制台，在左侧导航栏中单击服务列表，再在服务列表页面选择调用者列表，查看调用者列表。

   可以看到包含了`com.alibaba.sae.boot.IHelloService`，且可以查看该服务的服务分组和调用者 IP。

步骤3：结果验证

- 本地验证。

  ```
  curl http://localhost:17080/sayHello/SAE

  Hello, SAE (from Dubbo with Spring Boot)
  ```

- 在 SAE 中验证。

  ```
  curl http://localhost:8080/sayHello/SAE

  Hello, SAE (from Dubbo with Spring Boot)
  ```

步骤4：部署到 SAE

1. 分别在`DubboProvider`和`DubboConsumer`的`pom.xml`文件中添加如下配置，然后执行 `mvn clean package` 将本地的程序打成可执行的 JAR 包。

```
<build>
   <plugins>
       <plugin>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-maven-plugin</artifactId>
           <executions>
               <execution>
                   <goals>
                       <goal>repackage</goal>
                   </goals>
               </execution>
           </executions>
       </plugin>
```

```
    </plugins>
</build>
```

2. **根据您要部署的集群类型，参考应用部署概述部署应用。**

3 应用迁移

3.1 应用迁移概述

如果您的应用已经部署到生产环境并处于正常运行状态，为了保持业务不中断运行，并且不发生数据丢失，您可以采用平滑迁移的方式将应用迁移至 SAE。

迁移到 SAE 的价值

- SAE 为应用部署提供了启动参数灵活配置、流程可视化、服务优雅上下线和分批发布等功能，让您的应用发布可配、可查、可控。
- SAE 提供了服务发现与配置管理功能，您无需再自行运维 Eureka、ZooKeeper、Consul 等中间件组件，可以直接使用 SAE 提供的商业版服务发现与配置管理。
- SAE 控制台提供了统一的服务治理，目前支持查询发布和消费的服务详情。
- SAE 提供了动态扩、缩容功能，可以根据流量高峰和低谷实时地为您的应用扩容和缩容。
- SAE 提供了高级监控功能，除了支持基本的实例信息查询外，还支持微服务调用链查询、系统调用拓扑图、慢 SQL 查询等高级监控功能。
- 对于Spring Cloud 框架应用 SAE 提供限流降级功能，保证您的应用高可用。
- 对于Spring Cloud 框架应用 SAE 提供了全链路灰度功能，满足您的应用在迭代、更新时通过灰度进行小规模验证的需求。

什么是平滑迁移

如果您的 Spring Cloud 集群及应用已经部署在生产环境并处于正常运行状态中，现场需要将集群迁移到 SAE 享受完整的 SAE 功能，在迁移过程中，业务需要平稳运行而不中断，而保证应用平台运行不中断迁移到 SAE 即为平滑迁移。

说明:

如果您的集群尚未在生产环境中运行，或者您可以接受停机迁移，则无须参考本文进行平滑迁移，可直接将应用在本地开发完再部署到 SAE，详情请参见如下章节内容。

- Spring Cloud 应用：将 Spring Cloud 应用托管到 SAE
- Dubbo 应用：将 Dubbo 应用托管到 SAE

迁移流程

1. （必选）迁移应用

   迁移的应用通常是无状态的，需要先进行应用迁移。

2. （可选）迁移 SLB 或修改域名配置

   在应用迁移完成后，您还需要迁移 SLB 或修改域名配置。

   · SLB

   - 如果您的应用在迁移之前已经使用 SLB，应用迁移后可以复用该 SLB。您可以根据您的实际需求选择绑定 SLB 的策略，详情请参见 SLB 绑定概述。
   - 如果您的应用在迁移之前没有使用 SLB，建议在迁移完入口应用（如上图所示的 API Gateway）后，为该应用创建并绑定一个新的 SLB。
   - 迁移应用的方案中，推荐使用双注册和双订阅方案，以节约ECS 成本。如果由于某种原因（如原 ECS 端口被占用）不能复用之前的 ECS，则需要采用切流迁移方案，添加新的

ECS用于应用迁移。在应用迁移完成后，依据迁移前应用是否使用SLB，选择复用 SLB 或创建 SLB 并绑定到迁移后应用。

- 域名
  - 如果迁移后的应用可以复用 SLB，则域名配置无需修改。
  - 如果迁移后的应用需要创建新的 SLB 并绑定，则需要在域名中添加新的 SLB 配置，详情请参见域名 DNS 修改，并删除原来不再使用的 SLB。

3. （可选）迁移存储和消息队列
   - 如果应用迁移前已经部署在阿里云上，同时存储和消息队列同样使用了阿里云相关产品（如 RDS、MQ 等），则应用迁移完成后，迁移前的存储和消息队列无需迁移。
   - 如果应用迁移前没有部署在阿里云上，请提交工单或联系 SAE 技术支持人员为您提供完整的上云及迁移到 SAE 方案。

本文以 Demo 应用演示平滑迁移。Demo下载 Provider Demo , Consumer Demo。

迁移方案

迁移应用有两种方案，切流迁移、双注册和双订阅迁移方案。两种方案均可保证应用正常运行不中断情况下完成平滑迁移。

> 说明:
> 本文将主要介绍双注册和双订阅方案。

- 切流迁移方案

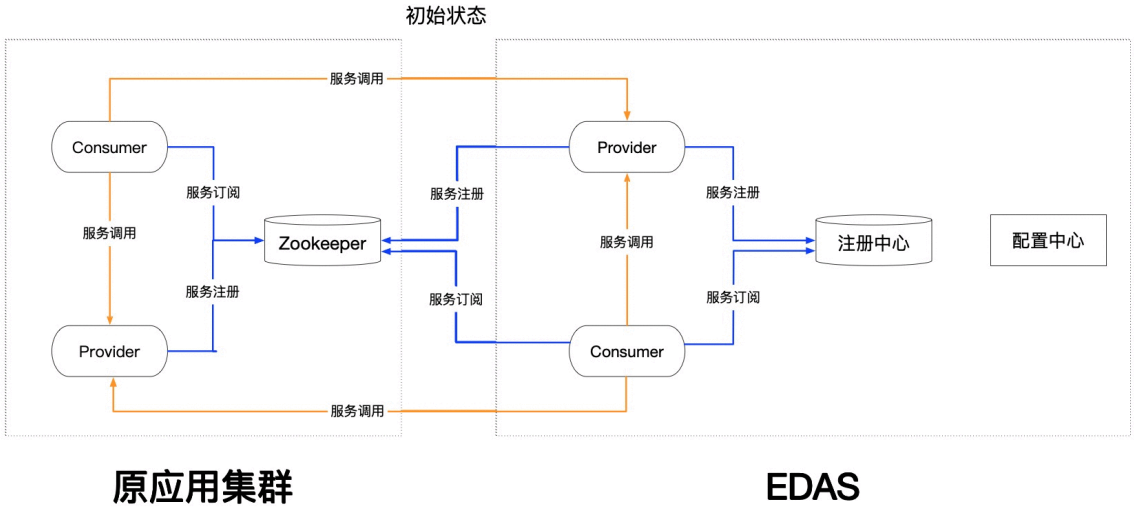
  使用 Dubbo 将原有的服务注册中心切换到 SAE ConfigServer，开发新的应用部署到 SAE，最后通过 SLB 和域名配置来进行切流。

  如果选择此方案，请参考微服务场景指引 开发应用。

- 双注册和双订阅迁移方案

  双注册和双订阅迁移方案是指在应用迁移时同时接入两个注册中心（原有注册中心和 SAE 注册中心）以保证已迁移的应用和未迁移的应用之间的相互调用。

  本方案实现架构图如下：



  - 已迁移的应用和未迁移的应用可以互相发现，从而实现互相调用，保证了业务的连续性。
  - 使用方式简单，仅需添加依赖，并修改极少的代码，可以实现双注册和双订阅。
  - 支持查看消费者服务调用列表的详情，实时地查看到迁移的进度。
  - 支持在不需要重启应用的情况下，动态地变更服务注册的策略和服务订阅的策略，只需要重启一次应用就可以完成迁移。

3.2 将 Dubbo 应用平滑迁移至 SAE

如果您的 Dubbo 应用已经部署在阿里云上，当前需要将应用迁移至SAE，请参考本文将应用平滑迁移到 SAE 中，并实现基本的服务注册与发现。如果您的 Dubbo 应用还未部署到阿里云，请提交工单或联系 SAE 技术支持人员为您提供完整的上云及迁移到 SAE 方案。

迁移流程

1. （必选）迁移应用

迁移的应用通常是无状态的，需要先进行应用迁移。

2. （可选）迁移 SLB 或修改域名配置

在应用迁移完成后，您还需要迁移 SLB 或修改域名配置。

- SLB
  - 如果您的应用在迁移之前已经使用 SLB，应用迁移后可以复用该 SLB。您可以根据您的实际需求选择绑定 SLB 的策略，详情请参见 SLB 绑定概述。
  - 如果您的应用在迁移之前没有使用 SLB，建议在迁移完入口应用（如上图所示的 API Gateway）后，为该应用创建并绑定一个新的 SLB。
  - 迁移应用的方案中，推荐使用双注册和双订阅方案，以节约ECS 成本。如果由于某种原因（如原 ECS 端口被占用）不能复用之前的 ECS，则需要采用切流迁移方案，添加新的 ECS用于应用迁移。在应用迁移完成后，依据迁移前应用是否使用SLB，选择复用 SLB 或创建 SLB 并绑定到迁移后应用。
- 域名
  - 如果迁移后的应用可以复用 SLB，则域名配置无需修改。
  - 如果迁移后的应用需要创建新的 SLB 并绑定，则需要在域名中添加新的 SLB 配置，详情请参见域名 DNS 修改，并删除原来不再使用的 SLB。

3. （可选）迁移存储和消息队列

- 如果应用迁移前已经部署在阿里云上，同时存储和消息队列同样使用了阿里云相关产品（如 RDS、MQ 等），则应用迁移完成后，迁移前的存储和消息队列无需迁移。
- 如果应用迁移前没有部署在阿里云上，请提交工单或联系 SAE 技术支持人员为您提供完整的上云及迁移到 SAE 方案。

本文以 Demo 应用演示平滑迁移。Demo下载 Provider Demo , Consumer Demo。

迁移方案

迁移应用有两种方案，切流迁移、双注册和双订阅迁移方案。两种方案均可保证应用正常运行不中断情况下完成平滑迁移。

说明:

本文将主要介绍双注册和双订阅方案。

- 切流迁移方案

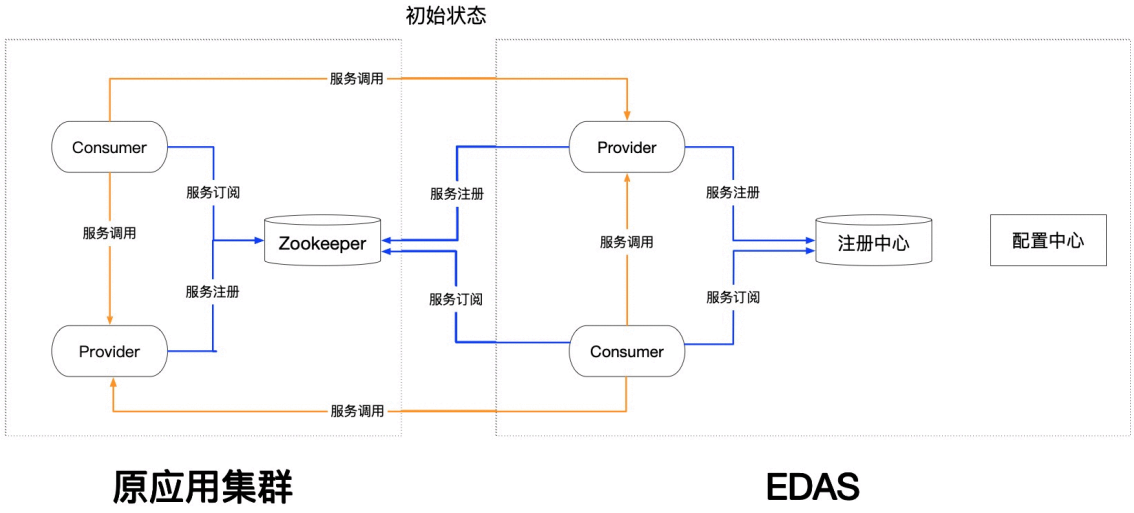
使用 Dubbo 将原有的服务注册中心同步到 SAE ConfigServer，开发全新的应用部署到 SAE ，通过 SLB 和域名配置进行切流。

选择此方案，请参考将 Dubbo 应用托管到 SAE 开发应用。

- 双注册和双订阅迁移方案

  双注册和双订阅迁移方案指在应用迁移时同时接入两个注册中心（原有注册中心和 SAE 注册中心），以保证已迁移的应用和未迁移的应用之间可相互调用。

  双注册和双订阅平滑迁移方案架构图如下：



  - 已迁移的应用和未迁移的应用之间可以互相发现，从而实现互相调用，保证了业务的连续性。
  - 使用方式简单，仅需要添加依赖，并修改极少代码，实现双注册和双订阅。
  - 支持查看消费者服务调用列表的详情，实时地查看到迁移的进度。
  - 支持在不重启应用的情况下，动态地变更服务注册的策略和服务订阅的策略，只需要重启一次应用就可以完成迁移。

迁移第一个应用

1. 选择迁移需求优先级最高的应用。

   建议从最下层 Provider 开始迁移。如果调用链路太复杂难分析，可以任意选一应用进行迁移。

2. 在应用程序中添加依赖并修改配置 (双注册、双订阅)。

   a. 在`pom.xml`文件中添加 `edas-dubbo-migration-bom` 依赖。

```
    <dependency>
            <groupId>com.alibaba.edas</groupId>
            <artifactId>edas-dubbo-migration-bom</artifactId>
            <version>2.6.5.1</version>
            <type>pom</type>
        </dependency>
```

b. 在 application.properties 中添加SAE注册中心的IP地址。

```
dubbo.registry.address = edas-migration://30.5.124.15:9999?service
-registry=edas://127.0.0.1:8080,zookeeper://172.31.20.219:2181&
reference-registry=zookeeper://172.31.20.219:2181&config-address=
127.0.0.1:8848
```

说明:

如果是非 Spring Boot 应用，在 dubbo.properties 或者对应的 Spring 配置文件中进行设置。

- edas-migration://30.5.124.15:9999

  多注册中心的头部可以不做修改，启动的时候，如果日志级别是为WARN 及以下，系统可能会上报WARN 的日志，因为 Dubbo 会对 IP 和端口进行校验，请忽略该日志。
- service-registry是服务的注册中心地址，支持服务的多注册中心，可以注入多个注册中心地址。每个注册中心均采用标准的 Dubbo 注册中心格式；多个用,分隔。 示例中 172.31.20.219 为ZooKeeper 地址，现场配置时请使用真实地址和端口。
- reference-registry是服务订阅的注册中心地址，支持多注册或者注册到未迁移前的注册中心。
- config-address是动态推送的地址。

c. 其他修改。

对于非 Spring Boot 的 Spring 应用，需要将com.alibaba.edas.dubbo.migration.controller.EdasDubboRegistryRest添加到您的扫描路径中。

3. 步骤三：本地验证

此处以动态配置的方式为例。

a. 准备工作

已完成 ZooKeeper下载、轻量配置中心配置、 Nacos下载及启动。

b. 检查服务是否成功注册。

- 登录轻量配置中心，在服务提供者列表中查看对应的服务。
- 登录 ZooKeeper，查看服务注册和消费信息。

c. （可选） 登录 Nacos，配置服务注册信息。

说明:

如果不需要动态配置的话，无需执行此步骤。

表 3-1: 配置服务注册信息

参数	配置及操作
DataId	设置为*dubbo.registry.config*。
Group	设置对应 Dubbo 应用的名称applicationName，如 *dubbo-migration-demo-server*。 配置信息是应用维度，所以应用名不能重复。

参数	配置及操作

配置内容	· 应用级别
	`dubbo.reference.registry=edas://127.0.0.1:8080 ##注册服务的注册中心`
	`dubbo.service.registry=edas://127.0.0.1:8080,zookeeper:127.0.0.1:2181 ##订阅服务的注册中心`
	· 实例 IP 级别
	`169.254.15.86.dubbo.reference.registry=edas://127.0.0.1:8080,zookeeper:127.0.0.1:2181`
	`169.254.15.86.dubbo.service.registry=edas://127.0.0.1:8080`
	集群验证时建议先验证实例 IP 级别，再验证整个应用验证。

d. 查看迁移后应用调用是否正常，查看注册中心的注册订阅关系。

- Spring Boot 1.x 版本：http://ip:port/dubboRegistry
- Spring Boot 2.x 版本：http://ip:port/actuator/dubboRegistry


```
{
  "dubbo.effective.reference.registry": [
    "edas://127.0.0.1:8080"
  ],
  "dubbo.orig.reference.registry": [
    "zookeeper://127.0.0.1:2181"
  ],
  "dubbo.orig.service.registry": [
    "edas://127.0.0.1:8080",
    "zookeeper://127.0.0.1:2181"
  ],
  "dubbo.effective.service.registry": [
    "edas://127.0.0.1:8080",
    "zookeeper://127.0.0.1:2181"
  ]
}
```

4. 步骤四：将应用部署到 SAE 中

   根据实际需求将应用部署到 ECS 集群或容器服务 Kubernetes 集群中，具体操作请参见部署应用概述。

   - 迁移前已使用 ECS，迁移后将 ECS 导入到 SAE 中，具体操作请参见导入 ECS 。

     > 注意:
     > 在导入 ECS 时如果系统提示需要转化后导入，请备份重要数据。

   - 如果需要创建新的 ECS、集群等资源，请在原有 VPC 内创建，保证迁移前后应用网络互通，实现应用平滑迁移。

     ECS、集群等资源创建，具体请参见创建资源。

   - 在数据库、缓存、消息队列等产品中为新 ECS 配置 IP 白名单等，确保应用所依赖的第三方组件正常访问。

5. 结果验证

a. 观察业务运行是否正常。

b. 查看服务订阅监控。

如果应用开启了 Spring Boot Actuator 监控功能，请访问 Actuator 查看此应用订阅的各服务的 RibbonServerList 信息。Actuator 地址如下：

- Spring Boot 1.x 版本：http://ip:port/dubboRegistry
- Spring Boot 2.x 版本：http://ip:port/actuator/dubboRegistry

`dubbo.orig.**`表示应用中配置的注册中心信息。

`dubbo.effective.**`表示生效的注册中心信息。

迁移其它所有应用

依照迁移第一个应用，依次将所有应用迁移到 SAE。

清理迁移配置

迁移完成后，删除原有的注册中心配置和迁移过程专用的依赖`edas-dubbo-migration-bom`。

修改对应的注册中心地址(即删除 ZooKeeper 的配置)，保证 Consumer、Provider 仅从 SAE 订阅。

- 方式一：动态配置

请参考本地验证中的方法进行修改。

- **方式二：手动修改**

  所有的应用修改完成后，修改应用的注册中心地址，将订阅的地址改为 *SAE ConfigServer*。

  ```
  dubbo.registry.address = edas-migration://30.5.124.15:9999?service
  -registry=edas://127.0.0.1:8080,zookeeper://172.31.20.219:2181&
  reference-registry=edas://127.0.0.1:8080&config-address=127.0.0.1:
  8848
  ```

  `reference-registry`的`zookeeper://172.31.20.219:2181`**改为**`edas://127.0.0.1:8080`**。修改完成之后，即可部署应用。**

  ```
  dubbo.registry.address = edas://127.0.0.1:8080
  ```

  说明:

  当应用迁移完成之后，如果不再使用`ZooKeeper`**，需要从注册中心配置中删除**`zookeeper://172.31.20.219:2181`

应用重启请选择业务量较小时间段分批进行。

迁移问题咨询

迁移过程中遇到异常情况，申请加入钉钉群进行咨询。

说明:

为了更好的服务您，请申请时备注公司名与阿里云账号。

3.3 将 Spring Cloud 框架应用平滑迁移至 SAE

如果您的 Spring Cloud 集群（包含多个应用）已经部署在阿里云上，当前需要将应用迁移至SAE，请参考本文将应用平滑迁移到 SAE 中，并实现基本的服务注册与发现。如果您的 Spring Cloud 集群还未部署到阿里云，请提交工单或联系 SAE 技术支持人员为您提供完整的上云及迁移到 SAE 方案。

迁移流程

1. （必选）迁移应用

   迁移的应用通常是无状态的，需要先进行应用迁移。

2. （可选）迁移 SLB 或修改域名配置

   在应用迁移完成后，您还需要迁移 SLB 或修改域名配置。

   · SLB

   - 如果您的应用在迁移之前已经使用 SLB，应用迁移后可以复用该 SLB。您可以根据您的实际需求选择绑定 SLB 的策略，详情请参见 SLB 绑定概述。
   - 如果您的应用在迁移之前没有使用 SLB，建议在迁移完入口应用（如上图所示的 API Gateway）后，为该应用创建并绑定一个新的 SLB。
   - 迁移应用的方案中，推荐使用双注册和双订阅方案，以节约ECS 成本。如果由于某种原因（如原 ECS 端口被占用）不能复用之前的 ECS，则需要采用切流迁移方案，添加新的

ECS用于应用迁移。在应用迁移完成后，依据迁移前应用是否使用SLB，选择复用 SLB 或创建 SLB 并绑定到迁移后应用。

- 域名
  - 如果迁移后的应用可以复用 SLB，则域名配置无需修改。
  - 如果迁移后的应用需要创建新的 SLB 并绑定，则需要在域名中添加新的 SLB 配置，详情请参见域名 DNS 修改，并删除原来不再使用的 SLB。

3. （可选）迁移存储和消息队列
   - 如果应用迁移前已经部署在阿里云上，同时存储和消息队列同样使用了阿里云相关产品（如 RDS、MQ 等），则应用迁移完成后，迁移前的存储和消息队列无需迁移。
   - 如果应用迁移前没有部署在阿里云上，请提交工单或联系 SAE 技术支持人员为您提供完整的上云及迁移到 SAE 方案。

本文以 Demo 应用演示平滑迁移。Demo下载 Provider Demo , Consumer Demo。

迁移方案

迁移应用有两种方案，切流迁移、双注册和双订阅迁移方案。两种方案均可保证应用正常运行不中断情况下完成平滑迁移。

> 说明:
> 本文将主要介绍双注册和双订阅方案。

- 切流迁移方案

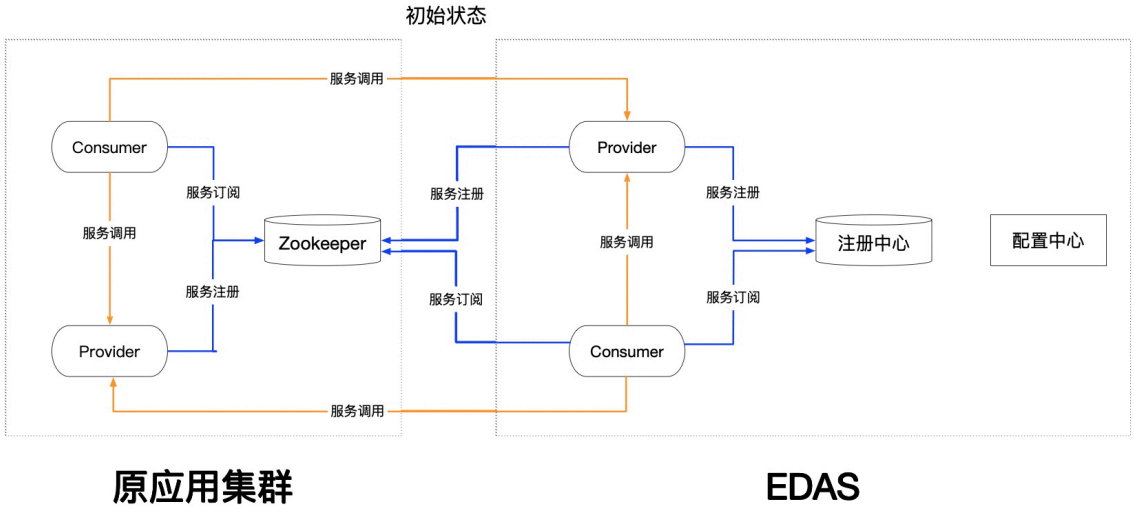
  使用 Spring Cloud Alibaba 将原有的服务注册中心切换到 Nacos。开发一套新的应用部署到 SAE，最后通过 SLB 和域名配置来进行切流。

  如果您选择此方案，那您可以参考将 Spring Cloud 应用托管到 SAE开发应用，不需要再阅读迁移应用的后续内容，只需要关注本文末尾的迁移风险点和技术支持。

- 双注册和双订阅迁移方案

  双注册和双订阅迁移方案指在应用迁移时同时接入两个注册中心（原有注册中心和 SAE 注册中心），以保证已迁移的应用和未迁移的应用之间可相互调用。

  双注册和双订阅平滑迁移方案架构图如下：



  - 已迁移的应用和未迁移的应用之间可以互相发现，从而实现互相调用，保证了业务的连续性。
  - 使用方式简单，仅需要添加依赖，并修改极少代码，实现双注册和双订阅。
  - 支持查看消费者服务调用列表的详情，实时地查看到迁移的进度。
  - 支持在不重启应用的情况下，动态地变更服务注册的策略和服务订阅的策略，只需要重启一次应用就可以完成迁移。

迁移第一个应用

1. 选择迁移需求优先级最高的应用。

   建议从最下层 Provider 开始迁移。如果调用链路太复杂难分析，可以任意选一应用进行迁移。

2. 步骤二：在应用程序中添加依赖并修改配置 。

   a. 在`pom.xml`文件中添加 `spring-cloud-starter-alibaba-nacos-discovery` 依赖。

```
<dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-alibaba-nacos-discovery</
artifactId>
     <version>{相应的版本}</version>
 </dependency>
```

   b. 在 `application.properties` 中添加 nacos-server 的IP地址。

```
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

c. Spring Cloud 默认依赖中只能引入一个注册中心，存在多个注册中心时，启动会异常。如果需要至支持多注册，添加依赖 `edas-sc-migration-starter`。

```
<dependency>
     <groupId>com.alibaba.edas</groupId>
     <artifactId>edas-sc-migration-starter</artifactId>
     <version>1.0.2</version>
 </dependency>
```

d. Ribbon 是实现负载均衡组件，应用从多个注册中心订阅服务，需要修改 Ribbon 配置。在应用启动的主类中，将 RibbonClients 默认配置修改为 `MigrationRibbonConfiguration`。

假设原有的应用主类启动代码如下：

```
@SpringBootApplication
 public class ConsumerApplication {
     public static void main(String[] args) {
         SpringApplication.run(ConsumerApplication.class, args);
     }
 }
```

修改后应用主类启动代码如下

```
@SpringBootApplication
 @RibbonClients(defaultConfiguration = MigrationRibbonConfi
guration.class)
 public class ConsumerApplication {
     public static void main(String[] args) {
         SpringApplication.run(ConsumerApplication.class, args);
     }
 }
```

说明：

在本地修改应用或者应用部署到 SAE 后，如果对应用有其它控制需求（如注册到哪些注册中心或从哪些注册中心订阅），可以通过 Spring Cloud Config 或 Nacos Config 进行动态的配置调整，无需重启应用，调整配置请参考动态调整服务注册和订阅方式。

通过 Spring Cloud Config 或 Nacos Config 进行动态配置调整，需要在其应用中添加配置管理依赖和修改配置，使用 Spring Cloud Config 请参考开源文档，使用 Nacos Config 请参考实现配置管理。

3. 步骤四：将应用部署到 SAE 中

根据实际需求将应用部署到 ECS 集群或容器服务 Kubernetes 集群中，具体操作请参见部署应用概述。

- 迁移前已使用 ECS，迁移后将 ECS 导入到 SAE 中，具体操作请参见导入 ECS 。

  
  注意：
  在导入 ECS 时如果系统提示需要转化后导入，请备份重要数据。

- 如果需要创建新的 ECS、集群等资源，请在原有 VPC 内创建，保证迁移前后应用网络互通，实现应用平滑迁移。

  ECS、集群等资源创建，具体请参见创建资源。

- 在数据库、缓存、消息队列等产品中为新 ECS 配置 IP 白名单等，确保应用所依赖的第三方组件正常访问。

4. 结果验证

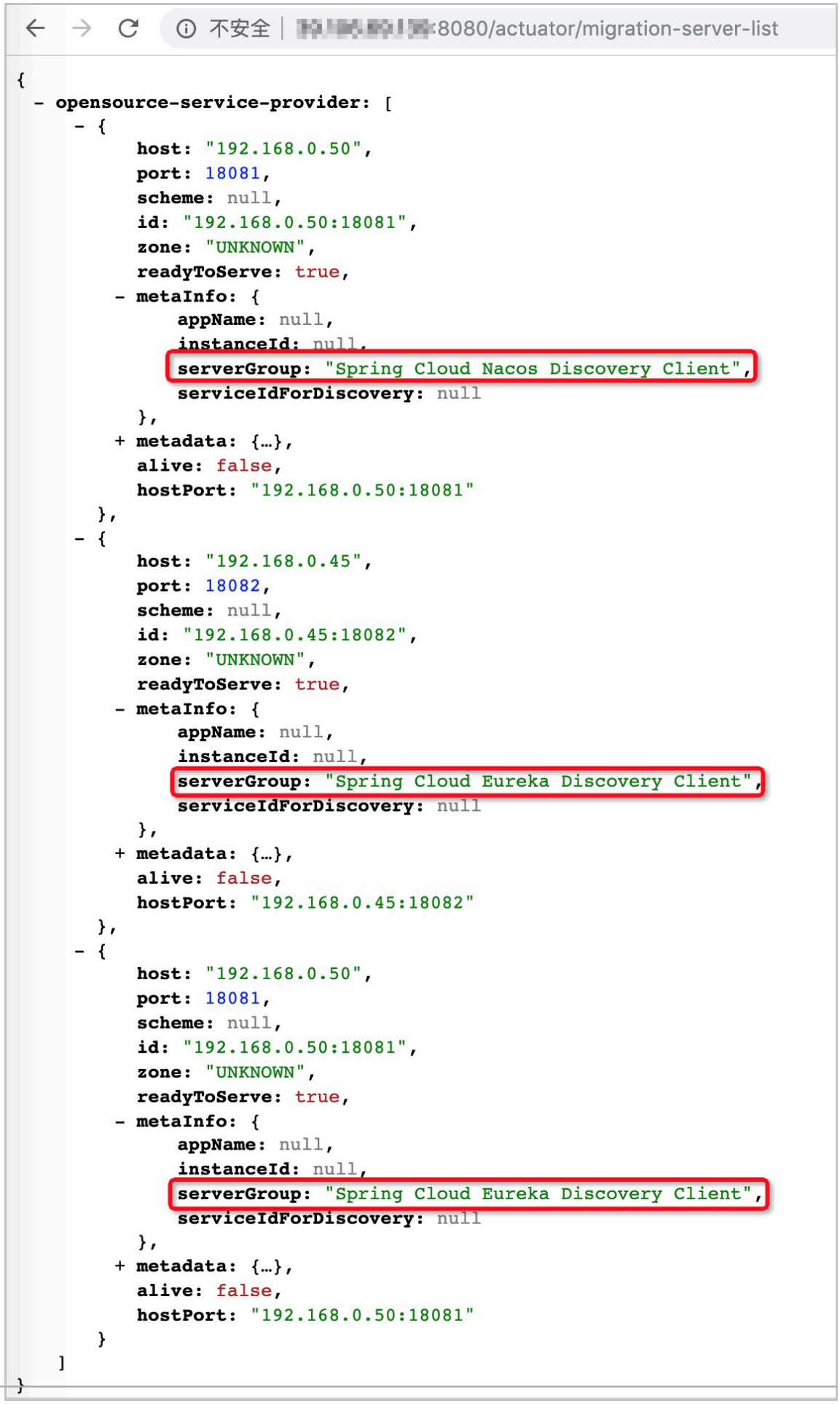
a. 观察业务运行是否正常。

b. 查看服务订阅监控。

如果应用开启了 Spring Boot Actuator 监控功能，请访问 Actuator 查看此应用订阅的各服务的 RibbonServerList 信息。Actuator 地址如下：

- Spring Boot 1.x 版本：http://ip:port/dubboRegistry
- Spring Boot 2.x 版本：http://ip:port/actuator/dubboRegistry

metaInfo 中 serverGroup 字段表示此节点的服务注册中心。


不安全 | [redacted]:8080/actuator/migration-server-list
{
- opensource-service-provider: [
- {
host: "192.168.0.50",
port: 18081,
scheme: null,
id: "192.168.0.50:18081",
zone: "UNKNOWN",
readyToServe: true,
- metaInfo: {
appName: null,
instanceId: null,
serverGroup: "Spring Cloud Nacos Discovery Client",
serviceIdForDiscovery: null
},
+ metadata: {…},
alive: false,
hostPort: "192.168.0.50:18081"
},
- {
host: "192.168.0.45",
port: 18082,
scheme: null,
id: "192.168.0.45:18082",
zone: "UNKNOWN",
readyToServe: true,
- metaInfo: {
appName: null,
instanceId: null,
serverGroup: "Spring Cloud Eureka Discovery Client",
serviceIdForDiscovery: null
},
+ metadata: {…},
alive: false,
hostPort: "192.168.0.45:18082"
},
- {
host: "192.168.0.50",
port: 18081,
scheme: null,
id: "192.168.0.50:18081",
zone: "UNKNOWN",
readyToServe: true,
- metaInfo: {
appName: null,
instanceId: null,
serverGroup: "Spring Cloud Eureka Discovery Client",
serviceIdForDiscovery: null
},
+ metadata: {…},
alive: false,
hostPort: "192.168.0.50:18081"
}
]
}

迁移其它所有应用

依照迁移第一个应用，依次将所有应用迁移到 SAE。

清理迁移配置

迁移完成后，删除原有的注册中心配置和迁移过程专用的依赖`edas-sc-migration-starter`。

`edas-sc-migration-starter` 迁移专用的 starter，长期使用对业务的稳定性没有影响，对于 Ribbon 负载均衡实现有一定的局限性，建议在迁移完毕后删除，并在业务量较小的时间段内进行分批重启应用。

- 动态调整服务注册和订阅方式

  应用迁移过程中，可以通过 SAE 配置管理功能动态变更服务注册和订阅方式。

- 动态调整服务订阅

  系统默认订阅策略是从所有注册中心订阅，并对数据进行聚合。

  您可以通过 SAE 的配置管理修改`spring.cloud.edas.migration.subscribes`属性，选择具体的注册中心订阅数据。

  ```
  spring.cloud.edas.migration.subscribes=nacos,eureka # 同时从 Eureka 和 Nacos 订阅服务

  spring.cloud.edas.migration.subscribes=nacos         # 只从 Nacos 订阅服务
  ```

- 动态变更服务注册

  系统默认订阅策略是从所有注册中心订阅。

  您可以通过 SAE 的配置管理来调整服务注册中心。

  通过修改`spring.cloud.edas.migration.registry.excludes`属性关闭指定的注册中心。

  ```
  spring.cloud.edas.migration.registry.excludes=    #默认值为空，注册到所有的服务注册中心

  spring.cloud.edas.migration.registry.excludes=eureka    #关闭 Eureka 的注册

  spring.cloud.edas.migration.registry.excludes=nacos,eureka    #关闭 Nacos 和 Eureka 的注册
  ```

  应用运行时如需要动态修改服务注册策略，可使用 Spring Cloud 配置管理功能在运行时修改此属性。

迁移问题咨询

迁移过程中遇到异常情况，申请加入钉钉群进行咨询。

> **说明:**
> **为了更好的服务您，请申请时备注公司名与阿里云账号。**