ALIBABA CLOUD

阿里云

Serverless 应用引擎

快速入门

文档版本：20220704

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	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。	`switch {active|stand}`

目录

1.SAE新手指南

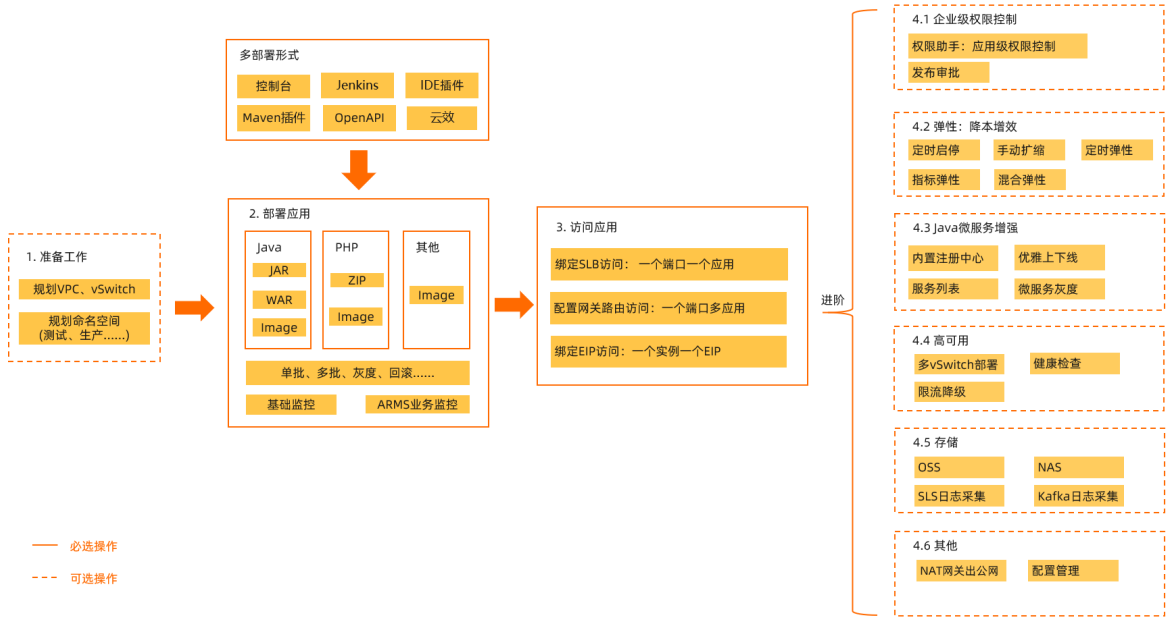
Serverless应用引擎SAE（Serverless App Engine）是面向应用的Serverless PaaS平台，能够帮助PaaS层用户免运维IaaS、按需使用、按量计费，做到低门槛微服务、PHP应用上云。本文介绍如何使用SAE，帮助您快速了解SAE以及各方面的最佳实践。

背景信息

如果您初次接触SAE，建议您观看入门视频，了解SAE及其基本操作。更多信息，请参见什么是Serverless应用引擎。

SAE使用流程图

SAE使用流程如下图所示。



1. 在首次部署SAE应用前，需要完成规划VPC、vSwitch、命名空间（区分测试、预发、生产等环境）的准备工作。

   具体操作，请参见准备工作。

2. 在SAE控制台部署应用。

   具体操作，请参见创建应用。除通过控制台部署应用外，SAE还支持通过Jenkins、IDE插件、Maven插件、OpenAPI和云效等多种方式来部署应用。

   > **说明** 如果您是第一次部署应用到SAE，则需要在SAE控制台创建应用。

3. 通过以下方式访问SAE应用。
   - 方式一：绑定SLB访问，一个端口一个应用。具体操作，请参见为应用绑定SLB。
   - 方式二：配置网关路由访问，一个端口多个应用。具体操作，请参见为应用配置网关路由（CLB）。
   - 方式三：绑定EIP访问，一个实例绑定一个EIP。具体操作，请参见配置弹性公网IP。
4. （可选）您可以为SAE应用配置更多进阶功能。
   - 企业级权限控制。
   - 弹性：降本增效。

- Java微服务增强。
- 高可用。
- 存储。
- 其他功能。

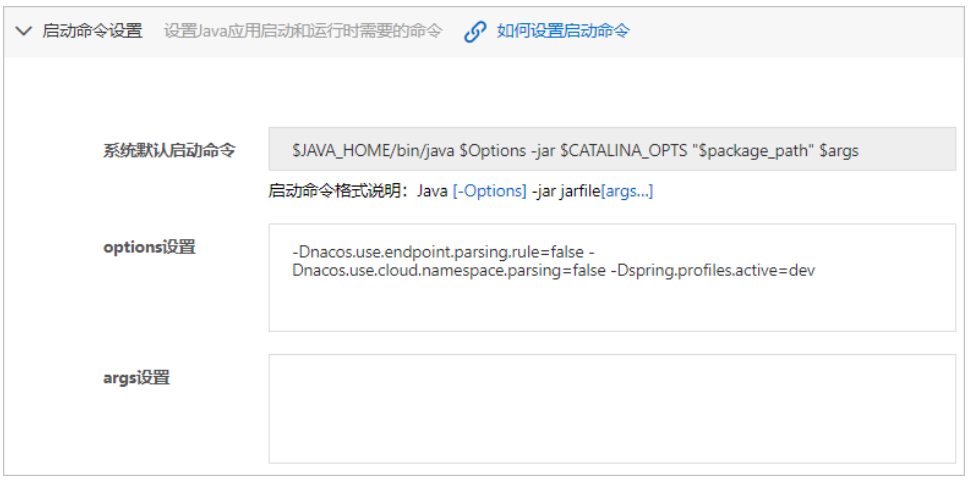
部署

SAE支持代码包部署和镜像部署，当前代码包支持Java的WAR包和JAR包，以及PHP ZIP包。在创建SAE应用时，需要自定义或者自动配置VPC、vSwitch和安全组等，并且需要指定实例规格（创建后可以修改规格）。本文以下列配置为例，介绍SAE的参考部署。

启动命令和参数

- 镜像部署
  启动命令以及相关参数可以直接写在Dockerfile内，同时也支持在SAE控制台覆盖启动命令，如下图所示。

- 代码包部署
  以JAR包部署为例，支持在SAE控制台配置启动相关参数，如下图所示。



数据库白名单

不同于ECS模式，SAE在容器形态下，每次部署应用时IP地址都有可能变化，所以需要在数据库的白名单内配置vSwitch的网段。具体信息，请参见应用访问阿里云数据库。

CI/CD

除控制台部署和API部署外，SAE和很多CI/CD工具进行了集成，典型的有云效和Jenkins，可以实现代码提交以后自动部署。具体信息，请参见CI/CD部署。

升级和回滚

SAE支持丰富的回滚策略，包括单批发布、分批发布、灰度发布和回退历史版本等。具体信息，请参见升级和回滚应用。

权限配置

SAE支持精细的权限控制，支持命名空间、应用和读写等粒度的权限控制，通过权限助手功能有效简化用户的配置过程。具体信息，请参见SAE权限助手。

其他

- 插件部署 ❯
- 端云互联 ❯
- Terraform ❯
- 环境变量 ❯
- Host绑定 ❯

网络

通过SAE部署应用后，您可以通过添加负载均衡SLB（Server Load Balancer）实现公网访问，也可以添加私网SLB实现同VPC内所有应用间的互相访问。针对访问公网的场景，SAE结合了NAT网关，既支持通过SLB+NAT实现入网和出网的网络方案，也支持在实例级别绑定EIP的网络方案。SAE提供了内置的Nacos注册中心，能够无缝迁移Spring Cloud和Dubbo应用。同时SAE联合MSE，增强了微服务能力。

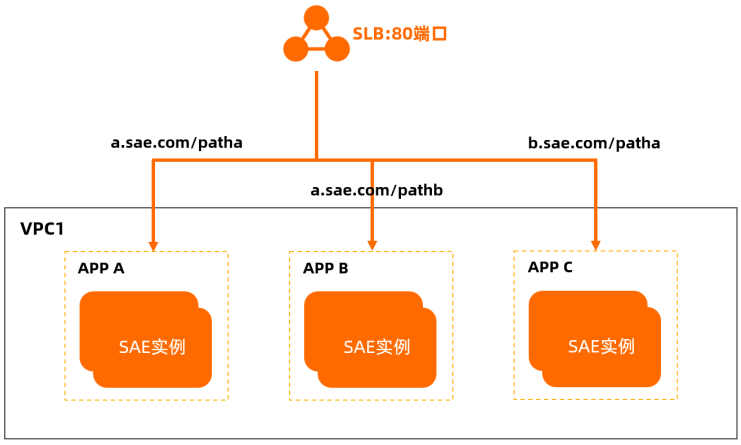
对于入口流量的请求，可以通过SAE绑定SLB。公网SLB适用于访问公网，私网SLB适用于提供一个稳定性的内网入口的场景。如下图所示。

入口流量：SLB

绑定SLB时，SAE支持TCP、HTTP和HTTPS协议，如无特殊需求，建议您使用HTTP或HTTPS协议。

入口流量：SLB（网关路由）

SAE不仅支持一个应用对应一个SLB端口，也支持一个SLB通过一个端口，根据不同的 `path` 路由到不同的应用。具体信息，请参见为应用配置网关路由（CLB）。

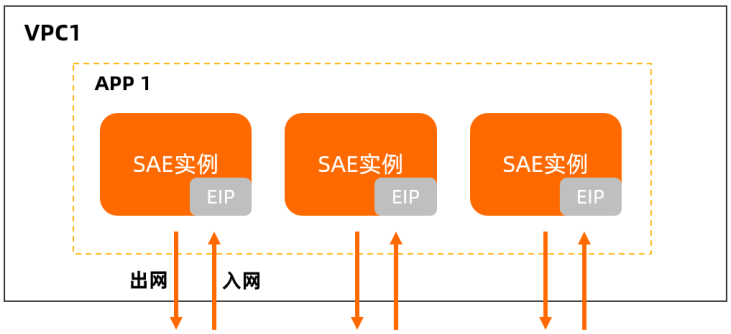


出公网：NAT

SAE结合NAT网关实现了访问公网的场景。具体信息，请参见部署在SAE上的应用如何访问公网。

EIP

与SLB+NAT的入网和出网的网络方案相比，SAE也支持在实例级别绑定EIP的方案。关于如何在SAE控制台绑定EIP，请参见配置弹性公网IP。



两者差异如下：

- 绑定级别不同：
  - NAT是VPC级别，一个VPC只需要配置一个NAT，其所有实例就都可以出公网。
  - SLB是应用级别，可以绑定多个应用 ，绑定以后这些应用也就都能被外网访问。
  - EIP是实例级别，10个实例就需要10个EIP，一旦实例绑定了EIP，就同时具备了出网和入网的能力。
- 适用场景不同：
  EIP适用于无设置自动弹性策略的需求，需要指定实例访问、指定实例长连接的场景。

微服务

- 注册中心
  SAE提供了内置的Nacos注册中心，无缝迁移Spring Cloud和Dubbo应用。具体信息，请参见使用Spring Cloud开发微服务应用并部署至SAE和将Dubbo应用托管到SAE。
  此外，SAE也支持自建注册中心或者使用MSE独立的注册中心，只需要在配置**启动命令设置**时，在启动参数中增加以下两个参数即可使用自定义的注册中心。参数如下所示。

```
-Dnacos.use.endpoint.parsing.rule=false -Dnacos.use.cloud.namespace.parsing=false
```

- 增强能力
  SAE联合MSE，提供的微服务增加能力如下所示：
  - 内置共享注册中心。
  - 服务列表。
  - 微服务优雅上下线。具体信息，请参见高可用。
  - 微服务灰度。

多语言

PHP运行时支持

SAE支持以下部署方式：

- 镜像：任意PHP架构应用。
- PHP ZIP包：任意PHP-FPM结合Nginx架构的在线应用。

SAE默认提供PHP运行时环境。更多信息，请参见PHP运行环境说明。

静态文件托管

借助NAS和OSS，可以独立托管静态文件，对运行过程中的代码、模板、上传的文件进行持久化存储，跨实例共享文件。

远程调试

借助SAE的不同功能，可以提供多种调试能力。

- PHP远程调试
  内置Xdebug插件，可开启远程调试。
- 文件下载
  可以通过Webshell登录实例，并借助SAE或OSS等功能下载文件。具体信息，请参见使用Webshell实现文件上传下载。
- 文件上传
  借助NAS和OSS，可方便地开发调试代码。

日志

SAE集成了SLS和Kafka日志采集。SAE实时日志功支持查看500行日志信息，如果您有更高的查阅需求，建议使用文件日志收集功能。SAE会将业务文件日志（容器内日志路径）、容器标准输出日志（stdout）收集并输入SLS或Kafka中，实现无限制行数查看日志、自行聚合分析日志，方便业务日志对接。

文件日志

SAE集成了SLS日志采集，只需在SAE控制台开启即可。与ECS时代手动维护采集的机器列表不同，在SAE配置好采集目录或者采集文件后，后续的每次部署、扩容，SAE都会自动和SLS对接日志采集，您即可在SLS控制台根据关键字来检索日志。具体信息，请参见设置日志收集至SLS。

日志源支持通配符，例如*/tmp/log/*.log*会采集*/tmp/log*目录以及子目录下所有以 `log` 结尾的文件。

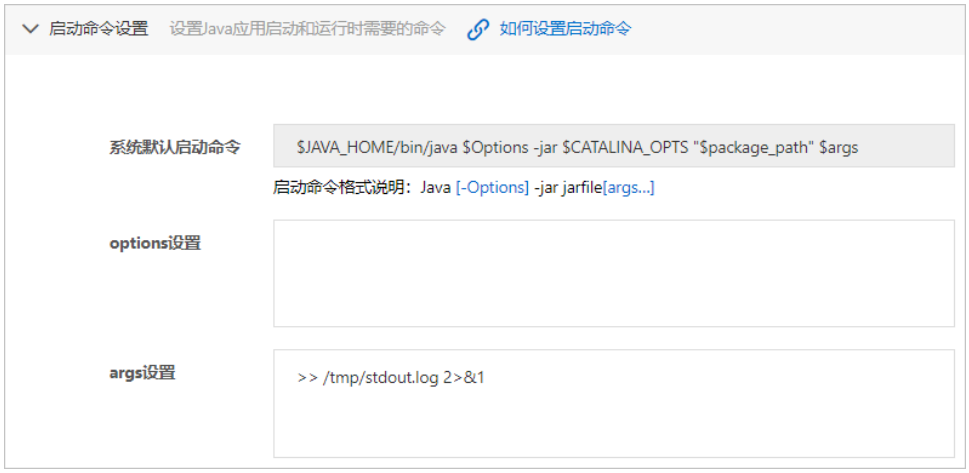
在不便使用SLS采集、RAM用户（子账号）无法查看SLS日志等情况下，您可以选择将日志导入Kafka。在此基础上，您可以结合自身的业务场景，将Kafka的数据投递到例如Elasticsearch等其他持久化库中，便于集中管理和分析日志。更多信息，请参见设置日志收集至Kafka和SAE日志导入Kafka最佳实践。

同时，您还可以通过环境变量来设置Logtail启动参数。具体信息，请参见设置环境变量提升Logtail采集性能。

实时日志

SAE会自动采集标准输出的日志，保留最新的500条。您可以在SAE控制台查看。具体信息，请参见日志管理。

如果您希望将标准输出内容也采集到SLS，可以先将标准输出到文件，再配置文件采集。参数如下图所示。



存储

SAE自带20 GB的系统盘存储，如果您有读写外部存储需求，建议使用NAS和OSS。诊断SAE应用包括常规检查和上传日志两种方式，其中上传日志不仅可以使用OSS，也可以使用SAE内置的一键上传与下载功能。

> **注意** 不建议将NAS和OSS用在日志的场景，对于日志场景建议使用SLS。具体信息，请参见设置日志收集至SLS。

NAS

SAE支持NAS存储功能，解决了应用实例数据持久化和实例间数据分发的问题。NAS存储只有挂载到ECS或者SAE才能访问。具体信息，请参见设置NAS存储。

OSS

OSS提供了便捷的工具以及控制台，支持可视化地管理Bucket。OSS适合读多写少的场景，例如挂载配置文件或者前端静态文件等。在SAE控制台部署应用时配置了OSS存储，您就可以通过OSS控制台访问数据。具体信息，请参见设置OSS存储。

> **注意** ossfs工具不能用在写日志的场景。具体信息，请参见ossfs概述。

上传下载文件

如果您希望将SAE内的文件下载到本地，您可以使用Webshell自带的文件上传下载功能。具体信息，请参见使用Webshell实现文件上传下载。

除使用NAS或者OSS存储外，您还可以使用ossutil工具。关于如何使用阿里云OSS服务进行日志上传下载，请参见通过常规检查诊断应用。

监控告警

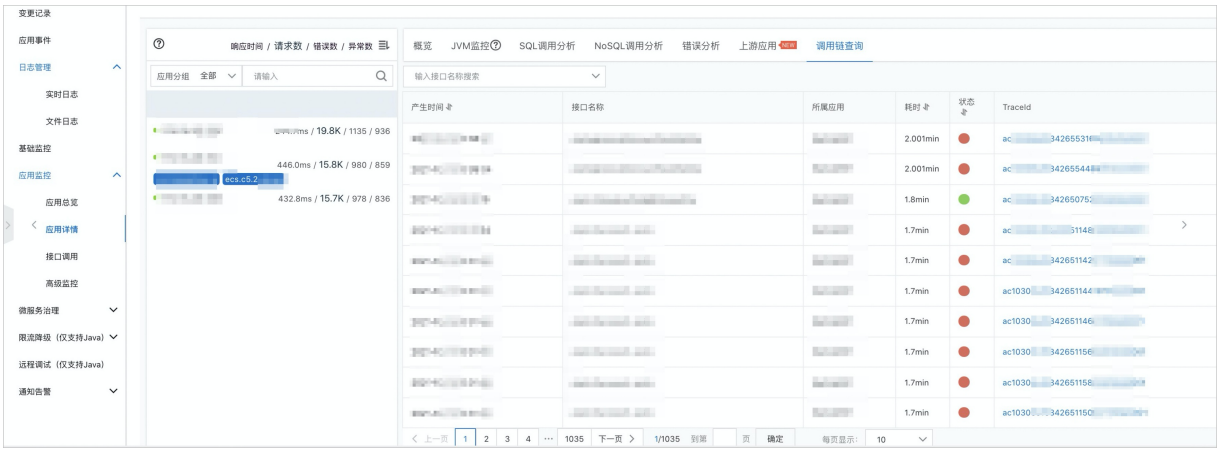
SAE内置了基础监控和ARMS业务监控（Java和PHP）。告警管理提供了可靠的告警收敛、通知、自动升级以及其他功能，帮助您快速检测和修复业务告警。

基础监控

基础监控包含CPU、Load、MEM、磁盘、网络和TCP连接等。具体信息，请参见基础监控。当前基础监控内置的是阿里云云监控产品，所以您也可以登录云监控控制台配置自定义的监控大盘。

业务监控

业务监控默认提供了ARMS基础版，可以查看应用大盘、JVM监控、接口QPS RT、线程池和调用链分析等。具体信息，请参见应用详情。



告警设置

SAE支持对以上各个监控项设置告警。具体信息，请参见应用监控报警。

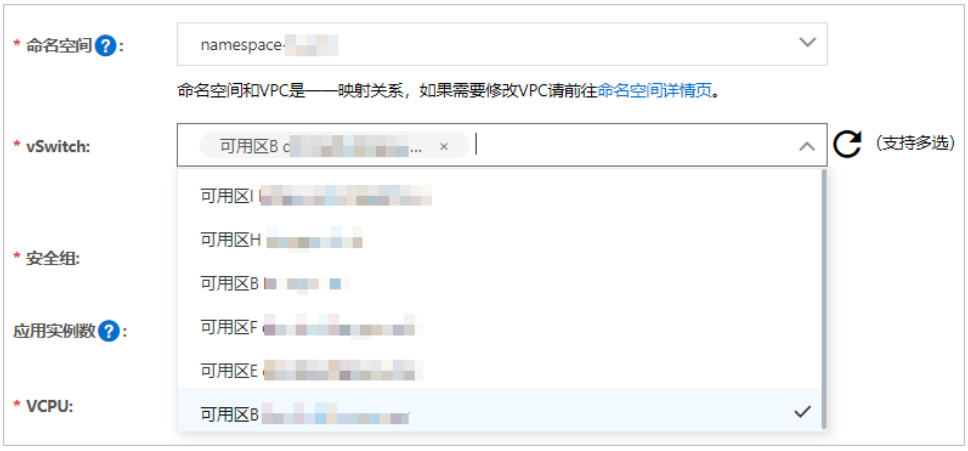
高可用

将应用部署在SAE后，针对流量的优雅上线与下线，您可以使用健康检查功能查看应用实例与业务运行是否正常，以便运行异常时定位问题。同时SAE支持部署多vSwitch以应对机房级别的故障，支持使用AHAS实现Java应用的限流降级，全面保障应用的可用性。

多vSwitch部署

为了应对机房级的故障，建议您对生产级别的SAE应用配置多vSwitch。您可以选择在创建应用时配置多vSwitch，或者在创建应用以后增加vSwitch。创建vSwitch时，建议您设计相对充足的IP地址数（建议100以上），如果IP地址不足则会出现无法创建或者弹性伸缩失败的情况。具体信息，请参见切换vSwitch。

- 创建时选择多vSwitch。



- 创建后增加vSwitch。

> 注意 增加vSwitch时，要注意数据库白名单。具体信息，请参见应用访问阿里云数据库。

优雅上下线

在SAE部署应用时，一般会经过先扩容再缩容的过程，但是对于流量的优雅上线与下线有两个关键步骤：

- 新扩容的实例是否已经就绪能够承接流量。
- 如何优雅销毁老实例。

SAE基于Kubernetes，提供了两种健康检查方式，包括应用实例存活检查（Liveness配置）和应用业务就绪检查（Readiness配置）。对于上述两个场景，在SAE中可以配置健康检查Readiness。Readiness探针会周期性地检测实例是否就绪，新实例就绪后才会接入流量。如果检测失败，则不会引入流量。开始销毁老实例前，会先从流量侧摘除，并且销毁实例前可以配置下线脚本以及等待时间。具体信息，请参见设置健康检查。

健康检查Liveness，也会周期性地检测。如果检测失败，会自动重启容器。该特性方便了异常场景的自动化运维，但是缺点是可能会丢失了现场，无法排查失败原因。您可以根据您的实际场景判断是否需要配置Liveness。

如果是微服务场景，因为注册中心的缓存，仅仅配置Readiness或者Liveness还不够，还需要配置微服务的优雅下线。具体信息，请参见设置微服务无损下线。同时，在生产环境中，可能会出现因使用自动弹性伸缩、回滚升级等能力，而导致短时间内服务不可用、业务监控大量报错等情况，这时候就需要配置微服务的优雅上线。具体信息，请参见设置微服务无损上线。

限流降级

对于大流量场景，SAE集成了AHAS限流降级。具体信息，请参见设置限流降级。

弹性（降本增效）

SAE提供了丰富的弹性策略，包括手动扩缩、定时弹性、指标弹性、混合弹性和定时启停。弹性作为云原生架构和应用的典型特点，从机器成本角度能够帮您降低成本，从运维角度也能够帮您提升运维效率。SAE提供了如下丰富的弹性策略。

手动扩缩

手动扩缩适用于人工运维的场景，相比于ECS相对较复杂和慢的扩缩；SAE扩缩基于容器镜像，可以快速实现扩容和缩容。具体信息，请参见手动扩缩。

定时弹性

定时弹性适用于流量可预期的场景。例如餐饮和教育行业，每天存在明确的早晚高峰，则可以配置在不同的时间段运行不同的实例数，使服务器资源尽量贴合实际业务流量。具体信息，请参见配置弹性伸缩策略。

指标弹性

指标弹性适用于流量相对不可预期的场景，当前支持CPU、MEM、TCP连接数、QPS和RT等指标。具体信息，请参见配置弹性伸缩策略。

混合弹性

混合弹性适用于同时有突发流量和定时弹性诉求的场景，例如互联网、教育和餐饮等行业，对已知的时段进行精细粒度的实例数调整。

例如在工作日弹性的实例最大值配置为 `max` ，最小值配置为 `min` ，但是实际上在周末又不需要保持 `min` 的实例数，则可以针对周末再配置一个实例数，减小 `min` 。具体信息，请参见配置弹性伸缩策略。

定时启停

定时启停功能可以实现按命名空间定时批量启停应用，例如定时启停开发环境或测试环境的全部应用。对于开发测试环境，假设每天只需在08:00～20:00时使用，其余时间处于闲置状态，则可以在SAE配置定时启停，以降低成本。具体信息，请参见管理定时启停规则。

最佳实践

针对各种业务需求，SAE提供了相关的最佳实践。除本文涉及到的弹性、网络、存储和应用访问阿里云等外，还包括镜像、应用加速、JVM参数配置等。更多常见场景，请参见SAE最佳实践。

2.准备工作

本文介绍您在使用Serverless应用引擎SAE（Serverless App Engine）托管应用之前需要完成的准备工作，包括开通服务、为RAM用户授权、创建命名空间和创建VPC。命名空间为您构建隔离的资源环境，VPC为您构建隔离的网络环境。

步骤一：开通服务并授权（阿里云账号）

使用SAE前，您需要开通SAE服务并对阿里云账号授权。SAE支持免费开通服务，开通后可以按照实际使用量进行计费。SAE计费方式分为按量付费和预付资源包两种，详细说明，请参见计费概述。

1. 登录SAE产品主页。
2. 单击**免费开通**。
3. 在**登录**页面输入阿里云账号和密码，单击**登录**。
4. 在**Serverless应用引擎**页面选中**Serverless应用引擎服务协议**，并单击**立即开通**。
   开通完成后，系统弹出**恭喜，开通成功！**对话框。
5. 单击**管理控制台**，在控制台首页弹出的**安全授权提示**对话框中，单击**立即授权**。
6. 在**云资源访问授权**页面单击**同意授权**。

> **说明**
> SAE支持SLR的自动创建。如果您未创建SLR，那么登录SAE控制台时会弹出**欢迎使用Serverless应用引擎SAE**对话框，单击**确认创建**即可完成SLR的创建。

（RAM用户必选）步骤二：为RAM用户授权

当您首次使用RAM用户登录SAE控制台时，SAE会弹出**欢迎使用Serverless应用引擎SAE**对话框提示您需要为RAM用户授权，使RAM用户可以访问相应的阿里云资源。具体操作，请参见为RAM用户授权。

步骤三：创建命名空间

命名空间以应用的服务调用与分布式配置推送为视角，提供逻辑隔离的运行环境。如果您有开发环境、测试环境和生产环境等场景，建议使用命名空间，将应用逻辑隔离，便于您管理应用及一键启停应用，提高您应用的安全性。

1. 登录SAE控制台。
2. 在左侧导航栏单击**命名空间**，在顶部菜单栏选择地域，单击**创建命名空间**。

3. 在**创建命名空间**面板，配置以下参数，单击**确定**。
   - **命名空间名称**：输入命名空间名称。
   - **命名空间ID**：命名空间ID的前缀已根据所选地域而确定，不可编辑；您只需设置自定义部分，自定义部分仅允许输入英文字母或数字。
   - **归属地域**：您选择的地域，不可更改。
   - **描述**：输入命名空间的描述信息，最多可以输入64个字符。

步骤四：创建VPC

VPC为您的应用提供了完全隔离的网络环境，处于同一VPC的应用可以互相访问，从网络层面提高您应用的安全性。创建VPC的相关操作请参见搭建IPv4专有网络。

> **说明**
> - 如果您尚未开通专有网络VPC服务，请依据页面提示开通VPC服务。
> - 创建VPC时选择的地域必须与步骤三：创建命名空间时选择的地域保持一致。

3.Web应用入门

3.1. 将Demo应用部署到SAE

本文以Java开发的Demo应用程序，采用WAR包部署方式，向您展示如何将应用部署到Serverless应用引擎SAE（Serverless App Engine），并通过绑定公网SLB，让您的应用可以被公网访问。

前提条件

1. 开通SAE服务
2. 创建安全组
3. 下载Demo应用

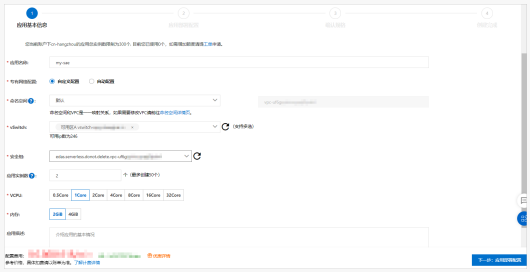
背景信息

该Demo应用为SAE欢迎网页程序，提供WAR包和JAR包两种部署方式。本文以WAR包方式部署为例。

操作步骤

步骤一：配置基本信息 | 步骤二：配置部署信息 | 步骤三：部署应用 | 步骤四：访问应用

1. 登录SAE控制台，在顶部菜单栏选择地域，在左侧导航栏单击**应用列表**。
   如果您是开通SAE后第一次登录控制台，请根据系统提示信息完成授权与手机验证。
2. 在**应用列表**页面单击**创建应用**，并在**应用基本信息**页签设置应用基本信息。



   - **应用名称**：输入my-sae。
   - **专有网络配置**：选择**自定义配置**。
   - **命名空间**：在下拉菜单中选择创建好的命名空间。
   - **vSwitch**：
     a. 单击右侧刷新图标，将已创建的vSwitch同步到SAE中。
        如果在下拉选项中显示所创建的vSwitch，则表示同步成功。
     b. 在下拉列表中选择创建好的**vSwitch**及**安全组**。
   - **应用实例数**：设置为2。
   - **VCPU**：选择1 Core。
   - **内存**：选择2 GiB。
3. 单击**下一步：应用部署配置**。

常见问题

- SAE资源如何收费？

SAE支持按量收费及预付费资源包。更多信息，请参见计费概述。

- 命名空间有什么用？
  命名空间是逻辑隔离的运行环境。从应用的服务调用与分布式配置推送的视角隔离不同的运行环境，如开发环境、测试环境和生产环境等。更多信息，请参见管理命名空间。
- 绑定SLB后应用仍无法被公网访问，如何处理？
  更多信息，请参见部署在SAE上的应用如何访问公网。
- 如何进行**环境变量设置**、**Hosts绑定设置**、**应用健康检查设置**、**日志收集服务**和**持久化存储**等高级配置？
  具体步骤，请参见高级配置。

更多信息

- 您在SAE部署完应用后，可以对应用进行更新、扩缩容、启停、删除应用等生命周期管理操作。具体操作，请参见管理应用生命周期。
- 您在SAE部署完应用后，可以对应用进行自动弹性伸缩、SLB绑定和批量启停等提升应用性能的操作。具体操作，请参见以下文档：
  - 绑定SLB
  - 配置弹性伸缩策略
  - 一键启停应用
  - 配置管理
  - 变更实例规格
- 您在SAE部署完应用后，还可以对应用进行日志管理、监控管理、应用事件查看和变更记录查看等聚焦应用运行状态的操作。具体操作，请参见以下文档：
  - 日志管理
  - 监控管理
  - 应用事件查看
  - 变更记录查看
  - 使用Webshell诊断应用

3.2. 将PHP应用部署到SAE

本文以PHP开发的Demo应用程序为例，采用ZIP包部署方式，向您展示如何将应用部署到Serverless应用引擎SAE（Serverless App Engine），并通过绑定公网SLB，让您的应用可以被公网访问。

前提条件

1. 开通SAE服务
2. 创建安全组

背景信息

本文的Demo应用为SAE欢迎网页程序，支持多种PHP框架。本文将演示PHP原生代码与部分框架的部署。

获取Demo

您可以按需选择以下Demo。

PHP原生

下载hello-sae-php.zip。解压后，可获得Nginx与PHP代码文件。具体说明，请参见PHP ZIP打包说明。

解压后的目录结构如下。

```
.
├── nginx
│   ├── default.conf
│   ├── fastcgi_params
│   └── root.dir
└── php
    ├── index.php
    └── phpinfo.php
```

Laravel 8.x

- 下载hello-sae-php-laravel.zip。解压后，可获得Nginx与PHP代码文件。
  PHP运行环境的具体说明，请参见PHP ZIP打包说明。
- 针对Laravel框架配置说明，参考如下。具体可查看*/nginx*与*/php*目录下的文件内容。
  - PHP应用代码，存放路径为*/home/admin/app/php/public*。
  - Nginx应用配置，存放路径为*/home/admin/app/nginx*，引用方式为 `include conf.d/fastcgi_params;` 。
  - Nginx系统配置，存放路径为*/etc/nginx*。

解压后的目录结构如下。

```
.
├── nginx                       # 存放路径/home/admin/app/nginx/
│   ├── default.conf
│   ├── fastcgi_params       # 引用方式include conf.d/fastcgi_params;
│   └── root.dir
├── php                         # 存放路径/home/admin/app/php
│   ├── README.md
│   ├── app
│   ├── artisan
│   ├── bootstrap
│   ├── composer.json
│   ├── composer.lock
│   ├── config
│   ├── database
│   ├── package.json
│   ├── phpunit.xml
│   ├── public               # Laravel框架默认目录，配置为 nginx root
│   ├── resources
│   ├── routes
│   ├── server.php
│   ├── storage
│   ├── tests
│   ├── vendor
│   └── webpack.mix.js
```

ThinkPHP 6

- 下载hello-sae-php-thinkphp.zip。解压后，可获得Nginx与PHP代码文件。
  PHP运行环境的具体说明，请参见PHP ZIP打包说明。

- 针对ThinkPHP框架配置说明，参考如下。具体可查看*/nginx*与*/php*目录下的文件内容。
  - PHP应用代码，存放路径为*/home/admin/app/php/public*。
  - PHP应用入口，存放路径为*/home/admin/app/php/public/index.php*。
  - Nginx应用配置，存放路径为*/home/admin/app/nginx*，引用方式为`include conf.d/fastcgi_params;`。
  - Nginx系统配置，存放路径为*/etc/nginx*。

解压后的目录结构如下。

```
.
├── nginx                        # 存放路径/home/admin/app/nginx/
│   ├── default.conf
│   ├── fastcgi_params       # 引用方式include conf.d/fastcgi_params;
│   └── root.dir
├── php                          # 存放路径/home/admin/app/php
│   ├── README.md
│   ├── app
│   ├── artisan
│   ├── bootstrap
│   ├── composer.json
│   ├── composer.lock
│   ├── config
│   ├── database
│   ├── package.json
│   ├── phpunit.xml
│   ├── public                 # Laravel框架默认目录，配置为nginx root
│   ├── resources
│   ├── routes
│   ├── server.php
│   ├── storage
│   ├── tests
│   ├── vendor
│   └── webpack.mix.js
```

WordPress 5

- 下载hello-sae-php-wordpress.zip。解压后，可获得Nginx与PHP代码文件。
  PHP运行环境的具体说明，请参见PHP ZIP打包说明。
- 针对WordPress框架配置说明，参考如下。具体可查看*/nginx*与*/php*目录下的文件内容。
  - PHP应用代码，存放路径为*/home/admin/app/php/public*。
  - PHP应用入口，存放路径为*/home/admin/app/php/public/index.php*。
  - Nginx应用配置，存放路径为*/home/admin/app/nginx*，引用方式为`include conf.d/fastcgi_params;`。
  - Nginx系统配置，存放路径为*/etc/nginx*。
- 若您为全新安装WordPress体验，且需要安装主题、插件或上传文件，您需要额外进行如下操作，对变更进行持久化管理。
  - 下载安装配置：可以通过Webshell下载保存*./php/wp-config.php*。具体说明，请参见Editing wp-config.php。
  - 安装SFTP：具体操作，请参见一键复制安装命令。

- 安装主题插件：安装后，需要下载保存*./php/wp-content/*目录内容并更新到ZIP包中，或配置OSS和NAS持久化存储到目录。具体操作，请参见通过日志上传下载诊断应用。

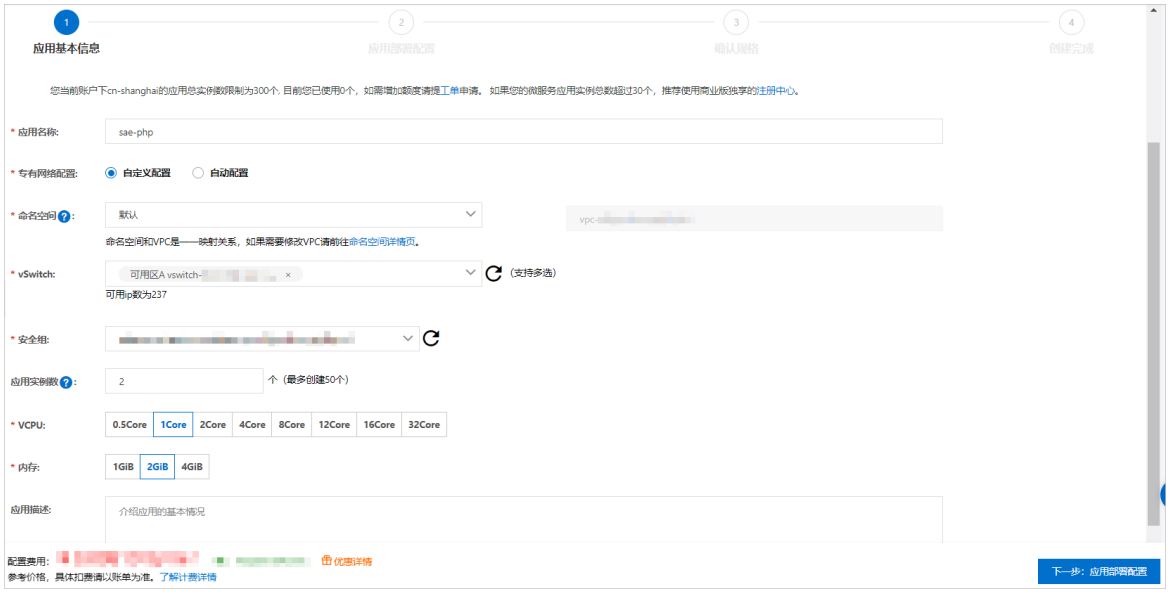
解压后的目录结构如下。

```
.
├── nginx                              # 存放路径/home/admin/app/nginx/
│   ├── default.conf                 # WordPress站点配置
│   ├── fastcgi_params               # 引用方式include conf.d/fastcgi_params;
│   ├── global
│   │   ├── restrictions.conf    # WordPress默认访问限制
│   │   └── wordpress.conf       # WordPress默认PHP访问规则
│   └── root.dir
├── php                                # 存放路径/home/admin/app/php
│   ├── index.php                    # WordPress框架默认入口
│   ├── license.txt
│   ├── readme.html
│   ├── wp-activate.php
│   ├── wp-admin                     # WordPress框架管理后台
│   ├── wp-blog-header.php
│   ├── wp-comments-post.php
│   ├── wp-config-sample.php
│   ├── wp-content
│   ├── wp-cron.php
│   ├── wp-includes
│   ├── wp-links-opml.php
│   ├── wp-load.php
│   ├── wp-login.php
│   ├── wp-mail.php
│   ├── wp-settings.php
│   ├── wp-signup.php
│   ├── wp-trackback.php
│   └── xmlrpc.php
```

操作步骤

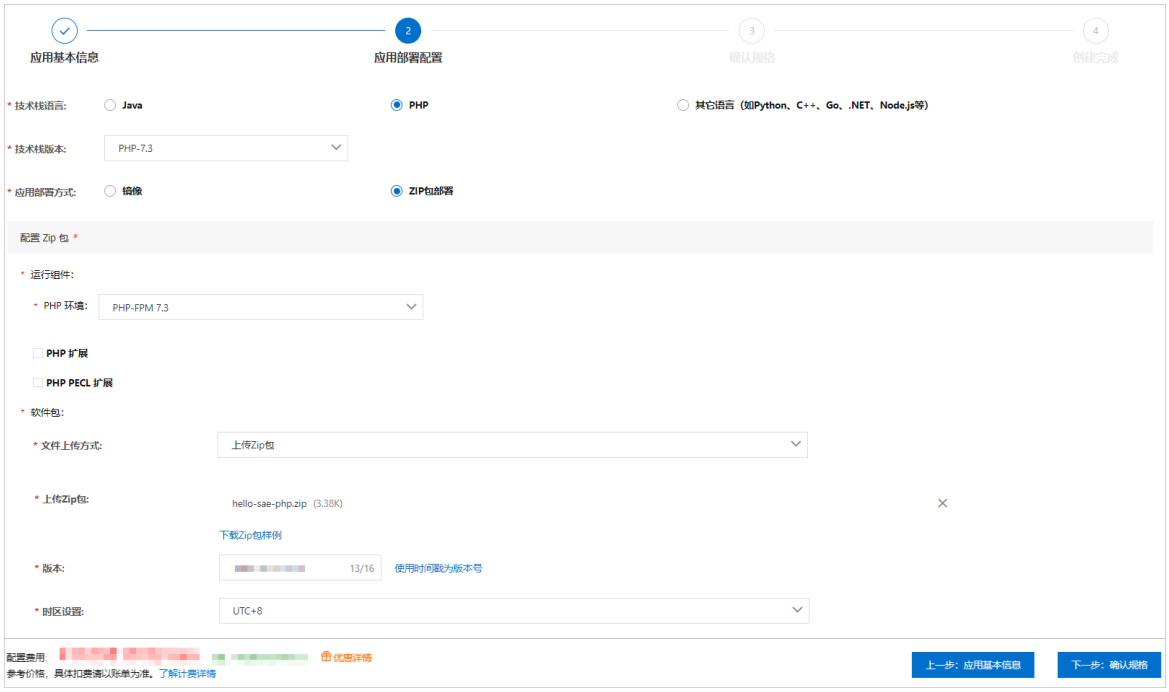
步骤一：创建PHP应用

1. 登录SAE控制台。
2. 在左侧导航栏单击**应用列表**，在顶部菜单栏选择地域，单击**创建应用**。
3. 在**应用基本信息**页签，配置相关参数，然后单击**下一步：应用部署配置**。



- **应用名称**：输入*sae-php*。
- **专有网络配置**：选择**自定义配置**。
- **命名空间**：在下拉列表选择命名空间，在右侧**请选择VPC**下拉列表选择已创建的VPC。
- **vSwitch**：在下拉列表选择vSwitch。
- **安全组**：在下拉列表选择安全组。
- **应用实例数**：设置为*2*。
- **VCPU**：选择**1 Core**。
- **内存**：选择**2 GiB**。

4. 在**应用部署配置**页面，配置相关参数，然后单击**下一步：确认规格**。



- **技术栈语言**：选择**PHP**。
- **技术栈版本**：在下拉列表选择**PHP-7.3**。

- **应用部署方式**：选择**ZIP包部署**。
- **配置Zip包**
  - **PHP环境**：在下拉列表选择**PHP-FPM 7.3**。
  - **文件上传方式**：在下拉列表选择**上传Zip包**。
  - **上传Zip包**：单击**选择文件**并上传您已获取的Demo。本文以PHP原生Demo为例。

其余参数保持默认设置。

5. 在**确认规格**页签，预览应用的部署信息，并单击**确认创建**。
6. 在**创建完成**页签，单击提示信息中的**应用详情页**查看所创建应用的详细信息。
   应用成功创建后，**实例部署信息**页签内实例的运行状态会显示为**Running**。

步骤二：通过公网访问应用

应用部署成功后，需为应用绑定SLB，以便公网通过SLB访问应用。SAE会自动帮您购买SLB服务，您仅需配置应用的监听端口；您也可以复用在

传统型负载均衡CLB
控制台购买的SLB。关于复用规则，请参见为应用绑定SLB。

1. 在**sae-php**应用的**基本信息**页面，单击**基本信息**页签。
2. 在**应用访问设置**区域，单击**添加公网SLB访问**。
3. 在**添加公网SLB访问**对话框，配置相关参数，然后单击**确定**。

   i. **请选择SLB**：在下拉列表选择**新建**。
   ii. 在**TCP协议**页签配置默认监听端口。
   - **SLB端口**：输入*80*。
   - **容器端口**：输入*80*。

   添加完成后，您可以在**基本信息**页面的**公网访问地址**区域看到该公网SLB的IP地址和端口。

4. 通过公网访问Demo应用。

   根据公网访问地址栏显示的公网SLB的IP和端口，在浏览器中以*http://slbip:port/*的格式输入地址并回车，可以看到Demo应用的首页。

常见问题

SAE如何支持高版本PHP部署？是否支持非PHP-FPM架构？

支持。您可以提交工单或加入钉钉群咨询灰度版本。

4.微服务应用入门

4.1. 微服务场景指引

Serverless应用引擎SAE（Serverless App Engine）支持原生Spring Cloud和Dubbo微服务框架的应用，您可以将基于原生Spring Cloud和Dubbo微服务框架开发的应用迁移、部署到SAE，进行微服务管理。

为什么使用SAE服务注册中心

- Spring Cloud
  SAE注册中心具备Spring Cloud Alibaba Nacos Discovery注册中心的所有功能。
  Spring Cloud Alibaba Nacos Discovery实现了Spring Cloud Registry标准接口，遵循Spring Cloud Registry标准规范。在实现服务注册与发现方面，与Eureka、Consul和ZooKeeper等组件相同。
  SAE服务注册中心可以完全代替Eureka、Consul、ZooKeeper和Redis，作为您的微服务应用的服务注册中心。与其相比，SAE还具有以下优势：
  - SAE服务注册中心为共享组件，为您节省了运维、部署ZooKeeper等组件的物理设备成本。
  - 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，为您提供了整套的微服务解决方案。
- Dubbo
  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，完整地支持Dubbo服务注册、路由规则和配置规则等功能。

  > 说明
  > - 关于Dubbo所提供的SPI标准的注册中心扩展内容，请参见注册中心扩展。
  > - 关于Dubbo服务注册，请参见注册和多注册中心。
  > - 关于Dubbo路由规则，请参见路由规则。
  > - 关于Dubbo配置规则，请参见配置规则。

您将应用部署到SAE时，SAE服务注册中心以高优先级自动设置Nacos Server服务端地址和服务端口，以及namespace、access-key、secret-key和context-path等信息，此外无需进行任何额外的配置。

原生Spring Cloud应用

- 如果您初次接触原生Spring Cloud应用，希望在SAE上部署原生Spring Cloud应用，您需要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。具体操作，请参见使用Spring Cloud开发微服务应用并部署至SAE。
- 如果您在本地开发了依赖Eureka、Consul、ZooKeeper和Redis等组件实现的服务注册与发现的Spring Cloud应用，希望将该应用部署至SAE，那么只需要将服务注册与发现的组件的依赖和配置替换成Spring Cloud Alibaba Nacos Discovery，无需修改任何业务代码，即可将应用部署到SAE进行微服务托管。具体操作，请参见将Spring Cloud应用托管到SAE。

原生Dubbo应用

- 如果您是初次接触原生Dubbo应用，希望在SAE上部署原生Dubbo应用，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。具体操作，请参见将Dubbo应用托管到SAE。
- 如果您在本地开发了依赖Eureka、Consul、ZooKeeper和Redis等组件实现的服务注册与发现的Dubbo应用，您希望将该应用部署至SAE，那么只需要将服务注册与发现的组件的依赖和配置替换成edas-dubbo-extension，无需修改任何业务代码，即可将应用部署到SAE进行微服务托管。具体操作，请参见将Dubbo

应用托管到SAE。

4.2. 将Spring Cloud应用托管到SAE

本文以包含服务提供者（Provider）和服务消费者（Consumer）的Spring Cloud微服务应用为例，指导您将原依赖Eureka、Consul、ZooKeeper等组件实现服务注册与发现的Spring Cloud应用部署到SAEServerless应用引擎SAE（Serverless App Engine）中，并实现应用的服务注册与发现，以及Consumer对Provider的调用。

前提条件

1. 下载并解压Nacos Server。
2. 进入*nacos/bin*目录，启动Nacos Server。
   - Linux、Unix、macOS系统：执行命令 `sh startup.sh -m standalone` 。
   - Windows系统：执行命令 `startup.cmd -m standalone` 。

   > **说明** `standalone` 表示单机模式运行，非集群模式。*startup.cmd*文件默认以集群模式启动，因此您在使用Windows系统时，如果直接双击执行*startup.cmd*文件会导致启动失败，此时需要在*startup.cmd*文件内设置 `MODE="standalone"` 。更多信息，请参见Nacos快速开始。

为什么托管到SAE

原依赖Eureka、Consul、ZooKeeper和Redis等组件实现服务注册与发现的Spring Cloud应用，如果需要部署至SAE，仅需将原服务注册与发现中心和配置中心替换为Alibaba Nacos Discovery，无需修改任何业务代码。

SAE服务注册中心具有Spring Cloud Alibaba Nacos Discovery所有功能，SAE服务注册中心可以完全代替Eureka、Consul、ZooKeeper和Redis等，作为您微服务应用的服务注册中心。

将Spring Cloud应用托管到SAE，您仅需关注Spring Cloud应用自身的逻辑，无需再关注注册中心和配置中心的搭建和维护，托管后还可以使用SAE提供的弹性伸缩、一键批量启停、应用监控等功能，大幅度降低开发和运维成本。

如果您选择商业版的服务注册中心，即使用MSE的Nacos作为服务注册中心，具体操作，请参见使用MSE的Nacos注册中心。

如果您选择使用自建的Nacos作为服务注册中心，具体操作，请参见自建Nacos服务注册中心。您需要确认以下内容：

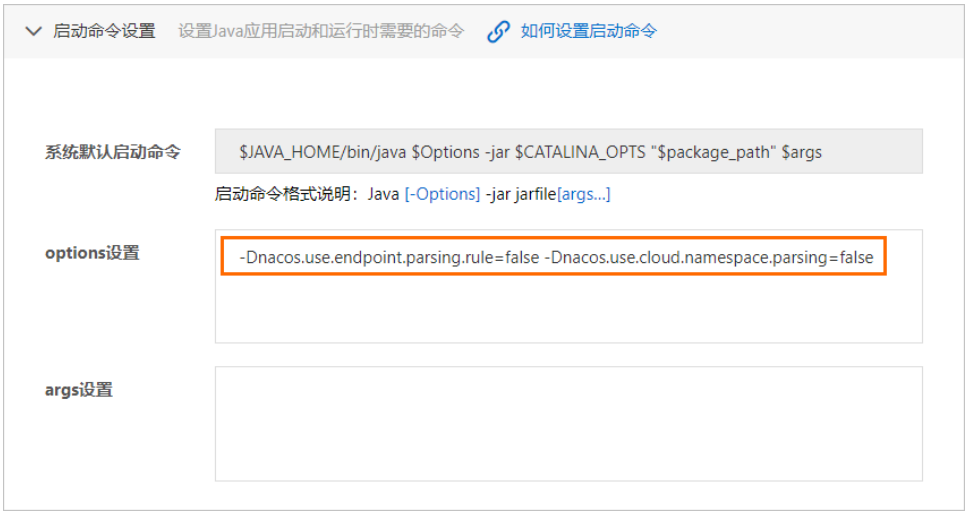
- 请确保SAE的网络与自建Nacos的网络互通。
- 在部署应用时建议使用镜像方式或者JAR包方式，并配置启动参数 `-Dnacos.use.endpoint.parsing.rule=false` 和 `-Dnacos.use.cloud.namespace.parsing=false` 。

  > **注意** 启动参数需要放在 `-jar` 之前，否则可能会导致无法使用非SAE自带的注册中心。

  - 如果采用镜像方式，请将 `-Dnacos.use.endpoint.parsing.rule=false` 和 `-Dnacos.use.cloud.namespace.parsing=false` 配置在镜像文件的程序启动命令中。关于Docker镜像制作方法，请参见制作Java镜像。
    示例代码如下：

    ```
    RUN echo 'eval exec java -Dnacos.use.endpoint.parsing.rule=false -Dnacos.use.cloud.namespace.parsing=false -jar $CATALINA_OPTS /home/admin/app/hello-edas-0.0.1-SNAPSHOT.jar'> /home/admin/start.sh && chmod +x /home/admin/start.sh
    ```

- 如果采用JAR包方式，请在控制台**启动命令设置**区域的**options设置**文本框输入 `-Dnacos.use.endpoint.parsing.rule=false -Dnacos.use.cloud.namespace.parsing=false` 。图示为**OpenJDK 8**运行环境下的Java应用。具体操作，请参见设置启动命令。



> **说明**
> 当您的微服务应用较多时，注册中心按推荐程度由高到低依次排序如下：
> - 商业版的服务注册中心（MSE）
> - 自建服务注册中心
> - SAE内置服务注册中心

步骤一：获取Demo

- 通过GitHub获取Demo。
  具体信息，请参见spring-cloud-simplest-demo。
- eureka-service-provider和eureka-service-consumer是SAE提供的微服务Demo应用程序包，二者均为已经接入Eureka的Spring Cloud应用。Consumer应用消费Provider应用提供的服务。
  - Provider应用：eureka-service-provider
  - Consumer应用：eureka-service-consumer

步骤二：修改Provider应用的服务注册与发现配置

将云原生的Provider应用托管到SAE中，需要在应用程序中修改pom依赖，并指定Nacos Server的IP地址。

> **说明** 以下情况下需要指定Nacos Server的IP地址：
> - 本地测试时，本地测试通过后再部署到SAE中。
> - SAE的服务注册中心为自建的Nacos。

1. 添加pom依赖。

   打开应用的*pom.xml*文件，将 `spring-cloud-starter-netflix-eureka-client` 替换成为 `spring-cloud-starter-alibaba-nacos-discovery` ，并设置Nacos Server的版本信息。
   替换前：

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

替换后：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    <version>2.1.1.RELEASE</version>
</dependency>
```

> **说明**
> - 示例中使用的版本是Spring Cloud Greenwich，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `2.1.1.RELEASE` 。
> - 如果您使用的是Spring Cloud Finchley版本，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `2.0.1.RELEASE` 。
> - 如果您使用的是Spring Cloud Edgware版本，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `1.5.1.RELEASE` 。

2. 指定Nacos Server的IP地址。

   打开*src\main\resources*路径下的*application.properties*文件，指定Nacos Server的IP地址。
   修改前：

```
spring.application.name=service-provider
server.port=18081
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8761/eureka/
```

   修改后：

```
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Nacos Server部署在其他机器上，则需要修改为对应的IP地址。如果有其他需求，请参见配置项参考在 `application.properties` 文件中增加所需配置。

3. 查询应用服务。

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

   ii. 登录本地启动的Nacos Server控制台 `127.0.0.1:8848/nacos` ，在左侧导航栏中选择**服务管理 > 服务列表**。

   > **说明** 本地Nacos Server控制台的默认用户名和密码均为nacos。

   如果**服务列表**中显示service-provider，且在详情中可以查询该服务的详情，那么表示服务注册成功。

步骤三：修改Consumer应用的服务注册与发现配置

将云原生的Consumer应用托管到SAE中，需要在应用程序中修改pom依赖，并指定Nacos Server的IP地址。

> **说明** 以下情况下需要指定Nacos Server的IP地址：
> - 本地测试时，本地测试通过后再部署到SAE中。
> - SAE的服务注册中心为自建的Nacos。

1. 添加pom依赖。

   打开应用的*pom.xml*文件，将 `spring-cloud-starter-netflix-eureka-client` 替换成为 `spring-cloud-starter-alibaba-nacos-discovery` ，并设置Nacos Server的版本信息。
   替换前：

   ```
   <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
   </dependency>
   ```

   替换后：

   ```
   <dependency>
       <groupId>com.alibaba.cloud</groupId>
       <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
       <version>2.1.1.RELEASE</version>
   </dependency>
   ```

   > **说明**
   > - 示例中使用的版本是Spring Cloud Greenwich，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `2.1.1.RELEASE` 。
   > - 如果您使用的是Spring Cloud Finchley版本，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `2.0.1.RELEASE` 。
   > - 如果您使用的是Spring Cloud Edgware版本，对应 `spring-cloud-starter-alibaba-nacos-discovery` 的版本为 `1.5.1.RELEASE` 。

2. 指定Nacos Server的IP地址。

   打开*src\main\resources*路径下的*application.properties*文件，指定Nacos Server的IP地址。
   修改前：

   ```
   spring.application.name=service-consumer
   server.port=18082
   eureka.client.serviceUrl.defaultZone=http://127.0.0.1:8761/eureka/
   ```

   修改后：

   ```
   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` 为Nacos Server的IP地址。如果Nacos Server部署在其他机器上，则需要修改为对应的IP地址。如果有其他需求，请参见配置项参考在 `application.properties` 文件中增加所需配置。

3. 查询应用服务。
   i. 执行 `nacos-service-consumer` 中 `ConsumerApplication` 的 `main` 函数，启动应用。
   ii. 登录本地启动的Nacos Server控制台 `127.0.0.1:8848/nacos` ，在左侧导航栏中选择**服务管理 > 服务列表**。

   > **说明** 本地Nacos Server控制台的默认用户名和密码均为nacos。

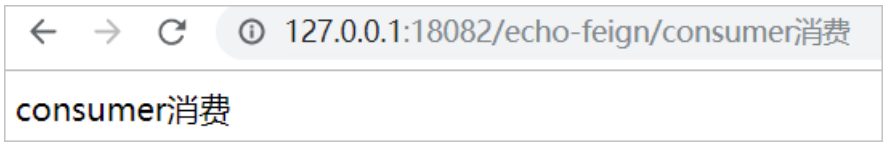
   如果**服务列表**中显示**service-consumer**，且在详情中可以查询该服务的详情，那么表示服务注册成功。

步骤四：查看Provider与Consumer的调用结果

在本地查看Consumer对Provider的服务调用结果。启动服务，执行 `IP+port/echo-rest/{自定义变量}` 或 `IP+port/echo-feign/{自定义变量}` 查看调用结果。

- Linux、Unix、macOS系统：执行 `curl http://127.0.0.1:18082/echo-rest/{自定义变量}` 或 `curl http://127.0.0.1:18082/echo-feign/{自定义变量}` 。
- Windows系统：在浏览器中输入 `http://127.0.0.1:18082/echo-rest/{自定义变量}` 或 `http://127.0.0.1:18082/echo-feign/{自定义变量}` 。

示例：以Windows系统为例，当显示以下结果时，表示Provider与Consumer业务调用成功。



步骤五：将应用部署到SAE

1. 在应用的*pom.xml*文件中添加应用程序的打包配置，添加完成后执行**mvn clean package**命令将本地的程序编译成可执行的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. 将编译好的Provider和Consumer应用包部署至SAE。具体操作，请参见部署微服务应用到SAE。

> 注意
>
> - SAE暂不支持创建空应用，因此第一次部署须在控制台完成。
> - 如果使用JAR包部署，在**应用部署配置**页签的**应用运行环境**须选择标准Java应用运行环境。
> - 如果使用WAR包部署，在**应用部署配置**页签的**应用运行环境**须选择apache-tomcat-XXX。

当您将应用部署到SAE时，SAE服务注册中心以高优先级自动设置Nacos Server服务端地址和服务端口，以及命名空间、AccessKey ID、AccessKey Secret等信息，您无需进行任何额外的配置。对于原有的配置内容您可以保留或删除。

步骤六：结果验证

1. 为Consumer应用绑定公网SLB，并在浏览器键入所设置的公网访问地址，进入应用首页。具体操作，请参见为应用绑定SLB。
2. 在应用首页发起调用请求，然后登录SAE控制台，在左侧导航栏单击**应用列表**。
3. 在顶部菜单栏，选择地域。
4. 在**应用列表**页面选择Consumer应用，进入**应用详情**页面。
5. 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。
   如果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}	当前服务的名称。
网卡名	spring.cloud.nacos.discovery.network-interface	无	当IP地址未配置时，注册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IP地址。
注册的IP地址	spring.cloud.nacos.discovery.ip	无	高优先级。
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	不同环境的注册逻辑隔离，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，您可以根据自己需要自定义和服务相关的元数据信息。

配置项	Key	默认值	说明
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获取服务端地址，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。

更多关于Spring Cloud Alibaba Nacos Discovery的信息，请参见开源版本的Spring Cloud Alibaba Nacos Discovery文档。

更多信息

- 您在SAE部署完应用后，可以对应用进行更新、扩缩容、启停、删除应用等生命周期管理操作。具体操作，请参见管理应用生命周期。
- 您在SAE部署完应用后，可以对应用进行自动弹性伸缩、SLB绑定和批量启停等提升应用性能的操作。具体操作，请参见以下文档：
  - 绑定SLB
  - 配置弹性伸缩策略
  - 一键启停应用
  - 配置管理
  - 变更实例规格
- 您在SAE部署完应用后，还可以对应用进行日志管理、监控管理、应用事件查看和变更记录查看等聚焦应用运行状态的操作。具体操作，请参见以下文档：
  - 日志管理
  - 监控管理
  - 应用事件查看
  - 变更记录查看
  - 使用Webshell诊断应用

4.3. 将Dubbo应用托管到SAE

本文以包含服务提供者Provider和服务消费者Consumer的Dubbo微服务应用为例，介绍如何在本地通过XML配置的方式，开发Dubbo微服务示例应用，并部署到SAE。

为什么托管到SAE

将Dubbo应用托管到SAE，您仅需关注Dubbo应用自身的逻辑，无需再关注注册中心和配置中心搭建和维护，托管后还可以使用SAE提供的弹性伸缩、一键启停、监控等功能，大大降低开发和运维成本。

当您的微服务应用较多时，注册中心按推荐程度由高到低依次排序如下：

- 商业版的服务注册中心（MSE）
- 自建服务注册中心
- SAE内置服务注册中心

如果您选择商业版的服务注册中心，即使用MSE的Nacos作为服务注册中心，具体操作，请参见使用MSE的Nacos注册中心。

如果您选择使用自建Nacos作为服务注册中心，具体操作，请参见自建Nacos服务注册中心。您需要确认以下内容：

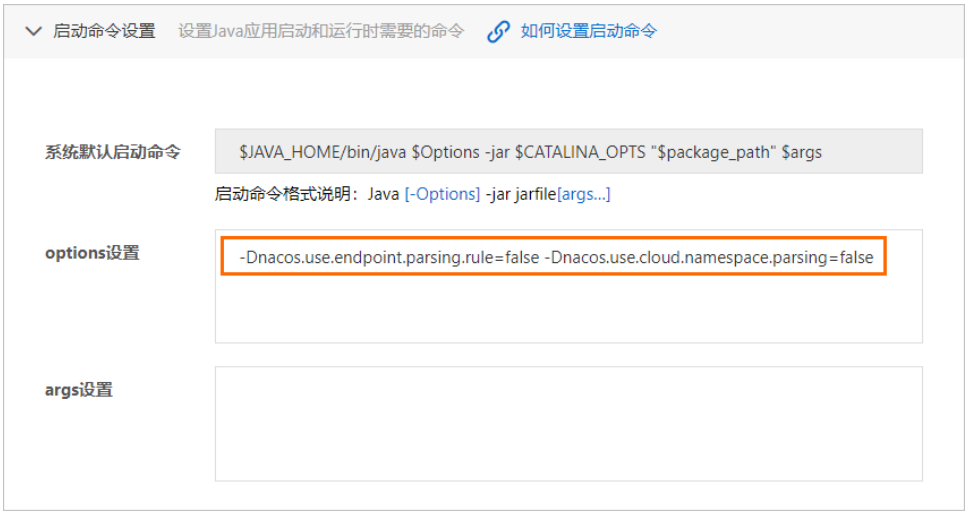
- 请确保SAE的网络与自建Nacos的网络互通。
- 在部署应用时建议使用镜像方式或者JAR包方式，并配置启动参数 `-Dnacos.use.endpoint.parsing.rule=false` 和 `-Dnacos.use.cloud.namespace.parsing=false` 。

  > **注意** 启动参数需要放在 `-jar` 之前，否则可能会导致无法使用非SAE自带的注册中心。

  - 如果采用镜像方式，请将 `-Dnacos.use.endpoint.parsing.rule=false` 和 `-Dnacos.use.cloud.namespace.parsing=false` 配置在镜像文件的程序启动命令中。关于Docker镜像制作方法，请参见制作Java镜像。
    示例代码如下：

    ```
    RUN echo 'eval exec java -Dnacos.use.endpoint.parsing.rule=false -Dnacos.use.cloud.namespace.parsing=false -jar $CATALINA_OPTS /home/admin/app/hello-edas-0.0.1-SNAPSHOT.jar'> /home/admin/start.sh && chmod +x /home/admin/start.sh
    ```

  - 如果采用JAR包方式，请在控制台**启动命令设置**区域的**options设置**文本框输入 `-Dnacos.use.endpoint.parsing.rule=false -Dnacos.use.cloud.namespace.parsing=false` 。图示为OpenJDK 8运行环境下的Java应用。具体操作，请参见设置启动命令。



> **说明**
> 当您的微服务应用较多时，注册中心按推荐程度由高到低依次排序如下：
> - 商业版的服务注册中心（MSE）
> - 自建服务注册中心
> - SAE内置服务注册中心

准备工作

在开始开发前，请确保您已经完成以下工作：

- 下载Maven并设置环境变量。

- 下载最新版本的Nacos Server。
- 按以下步骤启动Nacos Server。
  （可选）
  i. 解压下载的Nacos Server压缩包
  ii. 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。

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Nacos。

1. 创建Maven项目并引入依赖。
   i. 使用IDE（如IntelliJ IDEA或Eclipse）创建一个Maven项目。
   ii. 在 `pom.xml` 文件中添加dubbo、dubbo-registry-nacos和nacos-client依赖。

```
<dependencies>
    <dependency>
        <groupId>org.apache.dubbo</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.dubbo</groupId>
        <artifactId>dubbo-registry-nacos</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba.nacos</groupId>
        <artifactId>nacos-client</artifactId>
        <version>1.1.1</version>
    </dependency>
</dependencies>
```

2. 开发Dubbo服务提供者。

   Dubbo中服务都是以接口的形式提供的。
   i. 在 `src/main/java` 路径下创建一个package `com.alibaba.edas` 。
   ii. 在 `com.alibaba.edas` 下创建一个接口（interface） `IHelloService` ，里面包含一个 `SayHello` 方法。

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

iii. 在 `com.alibaba.edas` 下创建一个类 `IHelloServiceImpl`，实现此接口。

```
package com.alibaba.edas;
public class IHelloServiceImpl implements IHelloService {
    public String sayHello(String str) {
        return "hello " + str;
    }
}
```

3. 配置Dubbo服务。

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

ii. 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:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfr
amework.org/schema/beans/spring-beans-4.3.xsd
http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd
">
```

iii. 在 `provider.xml` 中将接口和实现类暴露成Dubbo服务。

```
<dubbo:application name="demo-provider"/>
<dubbo:protocol name="dubbo" port="28082"/>
<dubbo:service interface="com.alibaba.edas.IHelloService" ref="helloService"/>
<bean id="helloService" class="com.alibaba.edas.IHelloServiceImpl"/>
```

iv. 在 `provider.xml` 中将注册中心指定为本地启动的Nacos Server。

```
<dubbo:registry address="nacos://127.0.0.1:8848" />
```

- `127.0.0.1` 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地址。当将应用部署到SAE后，无需做任何修改，注册中心会替换成SAE上的注册中心的地址。
- `8848` 为Nacos Server的端口号，不可修改。

4. 启动服务。

i. 在 `com.alibaba.edas` 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服务暴露。

```
package com.alibaba.edas;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Provider {
    public static void main(String[] args) throws Exception {
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext
(new String[] {"provider.xml"});
        context.start();
        System.in.read();
    }
}
```

ii. 执行Provider的main函数，启动服务。

5. 登录Nacos控制台 `http://127.0.0.1:8848` ，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.edas.IHelloService` ，且可以查询该服务的**服务分组**和**提供者IP**。

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。

   i. 使用IDE（如IntellIJ IDEA或Eclipse）创建一个Maven项目。

   ii. 在 `pom.xml` 文件中添加dubbo、dubbo-registry-nacos和nacos-client依赖。

```
<dependencies>
    <dependency>
        <groupId>org.apache.dubbo</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.dubbo</groupId>
        <artifactId>dubbo-registry-nacos</artifactId>
        <version>2.7.3</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba.nacos</groupId>
        <artifactId>nacos-client</artifactId>
        <version>1.1.1</version>
    </dependency>
</dependencies>
```

2. 开发Dubbo服务提供者。

   Dubbo中服务都是以接口的形式提供的。

   i. 在 `src/main/java` 路径下创建Package，命名为 `com.alibaba.edas` 。

ii. 在 `com.alibaba.edas` 下创建一个接口（interface） `IHelloService` ，里面包含一个 `SayHell` o 方法。

> **说明** 通常是在一个单独的模块中定义接口，服务提供者和服务消费者都通过Maven依赖来引用此模块。本文档为了简便，服务提供者和服务消费者分别创建两个完全一模一样的接口，实际使用中不推荐这样使用。

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

3. 配置Dubbo服务。

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

ii. 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:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfr
amework.org/schema/beans/spring-beans-4.3.xsd
http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd
">
```

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

```
<dubbo:application name="demo-consumer"/>
<dubbo:registry address="nacos://127.0.0.1:8848"/>
<dubbo:reference id="helloService" interface="com.alibaba.edas.IHelloService"/>
```

4. 启动、验证服务。

i. 在 `com.alibaba.edas` 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.edas;
    import org.springframework.context.support.ClassPathXmlApplicationContext;
    import java.util.concurrent.TimeUnit;
    public class Consumer {
        public static void main(String[] args) throws Exception {
            ClassPathXmlApplicationContext context = new ClassPathXmlApplicationCon
text(new String[] {"consumer.xml"});
            context.start();
            while (true) {
                try {
                    TimeUnit.SECONDS.sleep(5);
                    IHelloService demoService = (IHelloService)context.getBean("hel
loService");
                    String result = demoService.sayHello("world");
                    System.out.println(result);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

ii. 执行Consumer的main函数，启动服务。

5. 验证创建结果。
   启动后，可以看到控制台不断地输出 `hello world` ，表明服务消费成功。
   登录Nacos控制台 `http://127.0.0.1:8848` ，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**。
   可以看到包含了 `com.alibaba.edas.IHelloService` ，且可以查看该服务的**服务分组**和**调用者IP**。

部署到SAE

1. 在*pom.xml*文件中添加应用编译配置，并执行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.edas.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.edas.Consumer</mainClass>
                   </configuration>
               </execution>
           </executions>
       </plugin>
 </plugins>
</build>
```

2. 将编译好的Provider和Consumer应用包部署至SAE。具体操作，请参见部署微服务应用到SAE。

更多信息

- 您在SAE部署完应用后，可以对应用进行更新、扩缩容、启停、删除应用等生命周期管理操作。具体操作，请参见管理应用生命周期。
- 您在SAE部署完应用后，可以对应用进行自动弹性伸缩、SLB绑定和批量启停等提升应用性能的操作。具体操作，请参见以下文档：
  - 绑定SLB

- 配置弹性伸缩策略
- 一键启停应用
- 配置管理
- 变更实例规格

- 您在SAE部署完应用后，还可以对应用进行日志管理、监控管理、应用事件查看和变更记录查看等聚焦应用运行状态的操作。具体操作，请参见以下文档：
  - 日志管理
  - 监控管理
  - 应用事件查看
  - 变更记录查看
  - 使用Webshell诊断应用

4.4. 开发HSF应用（Pandora Boot）

您可以使用Pandora Boot开发HSF应用，实现服务注册发现、异步调用，并完成单元测试。相比使用ali-tomcat部署HSF的WAR包，Pandora Boot部署的是JAR包。直接将HSF应用打包成FatJar，这更加符合微服务的风格，不需要依赖外置的ali-tomcat也使得应用的部署更加灵活。Pandora Boot可以认为是Spring Boot的增强。

前提条件

在开发应用前，您已经完成以下工作：

- 配置EDASSAE的私服地址和轻量级配置及注册中心
- 启动轻量级配置及注册中心

服务注册与发现

下面将介绍如何使用Pandora Boot开发应用（包括服务提供者和服务消费者）并实现服务注册与发现。

> **注意** 严禁在应用启动时调用HSF远程服务，否则会导致启动失败。

下载Demo源码。

使用Git克隆整个项目，并在*microservice-doc-demo/hsf-pandora-boot*文件夹内可以找到本文使用的示例工程。

1. 创建服务提供者。
   i. 创建命名为*hsf-pandora-boot-provider*的Maven工程。
   ii. 在*pom.xml*中引入需要的依赖。

```
<properties>
      <java.version>1.8</java.version>
      <spring-boot.version>2.1.6.RELEASE</spring-boot.version>
      <pandora-boot.version>2019-06-stable</pandora-boot.version>
  </properties>
  <dependencies>
      <dependency>
          <groupId>com.alibaba.boot</groupId>
          <artifactId>pandora-hsf-spring-boot-starter</artifactId>
      </dependency>
      <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>com.taobao.pandora</groupId>
                <artifactId>pandora-boot-starter-bom</artifactId>
                <version>${pandora-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>com.taobao.pandora</groupId>
                <artifactId>pandora-boot-maven-plugin</artifactId>
                <version>2.1.11.8</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

虽然HSF服务框架并不依赖于Web环境，但是在应用的生命周期过程中需要使用到Web相关的特性，所以需要添加 `spring-boot-starter-web` 的依赖。

`pandora-hsf-spring-boot-starter` 实现了HSF配置的自动装配。`pandora-boot-maven-plugin` 是Pandora Boot提供的Maven打包插件，可以将Pandora Boot HSF工程编译为可执行的FatJar，并在EDAS Container中部署运行。
`dependencyManagement` 中包含了 `spring-boot-dependencies` 和 `pandora-boot-starter-bom` 两个依赖，分别负责Spring Boot和Pandora Boot相关依赖的版本管理，设置之后，您的工程无需将parent设置为 `spring-boot-starter-parent` 。

iii. 定义服务接口，创建一个接口类 `com.alibaba.edas.HelloService` 。

HSF服务框架基于接口进行服务通信，当接口定义好之后，生产者将通过该接口实现具体的服务并发布，消费者也是基于此接口去订阅和消费服务。

```
public interface HelloService {
    String echo(String string);
}
```

接口com.alibaba.edas.HelloService提供了echo方法。

iv. 添加服务提供者的具体实现类EchoServiceImpl，并通过注解方式发布服务。

```
@HSFProvider(serviceInterface = HelloService.class, serviceVersion = "1.0.0")
public class HelloServiceImpl implements HelloService {
    @Override
    public String echo(String string) {
        return string;
    }
}
```

在HSF应用中，接口名和服务版本才能唯一确定一个服务，所以在注解HSFProvider中的需要添加接口名com.alibaba.edas.HelloService和服务版本 `1.0.0` 。

> **说明**
> - 注解中的配置拥有高优先级。
> - 如果在注解中没有配置，服务发布时会优先在*resources/application.properties*文件中查找这些属性的全局配置。
> - 如果注解和*resources/application.properties*文件中都没有配置，则会使用注解中的默认值。

v. 在*resources*目录下的*application.properties* 文件中配置应用名和监听端口号。

```
spring.application.name=hsf-pandora-boot-provider
server.port=8081
spring.hsf.version=1.0.0
spring.hsf.timeout=3000
```

> **说明** 建议将服务版本（ `spring.hsf.version` ）和服务超时（ `spring.hsf.timeout` ）都统一配置在*application.properties*中。

vi. 添加服务启动的main函数入口。

```
@SpringBootApplication
 public class HSFProviderApplication {
     public static void main(String[] args) {
         // 启动Pandora Boot用于加载Pandora容器。
         PandoraBootstrap.run(args);
         SpringApplication.run(HSFProviderApplication.class, args);
         // 标记服务启动完成，并设置线程wait。防止业务代码运行完毕退出后，导致容器退出。
         PandoraBootstrap.markStartupAndWait();
     }
 }
```

服务提供者属性列表

属性	是否必配	描述	类型	默认值
serviceInterface	是	服务对外提供的接口	Class	java.lang.Object
serviceVersion	否	服务的版本号	String	1.0.0.DAILY
serviceGroup	否	服务的组名	String	HSF
clientTimeout	否	该配置对接口中的所有方法生效，但是如果客户端通过methodSpecials属性对某方法配置了超时时间，则该方法的超时时间以客户端配置为准。其他方法不受影响，还是以服务端配置为准（单位ms）	int	-1
corePoolSize	否	单独针对这个服务设置最小活跃线程数，从公用线程池中划分出来	int	0
maxPoolSize	否	单独针对这个服务设置最大活跃线程数，从公用线程池中划分出来	int	0
delayedPublish	否	是否延迟发布	boolean	false
includeFilters	否	用户可选的自定义过滤器	String[]	空
enableTXC	否	是否开启分布式事务GTS	boolean	false

属性	是否必配	描述	类型	默认值
serializeType	否	服务接口序列化类型，hessian或者java	String	hessian
supportAsynCall	否	是否支持异步调用	String	false

服务创建及发布限制

名称	示例	限制大小	是否可调整
{服务名}:{版本号}	com.alibaba.edas.testcase.api.TestCase:1.0.0	最大192字节	否
组名	aliware	最大32字节	否
一个Pandora应用实例发布的服务数	N/A	最大800个	是，可在左侧导航栏单击**基本信息**在**基本信息**页签内的**应用设置**区域内，单击**JMV参数**右侧的**编辑**，然后在弹出的**应用设置**对话框中选择**自定义 > 自定义参数**，在输入框中添加 `-DCC.pubCountMax=1200` 属性参数（该参数值可根据应用实际发布的服务数调整）

2. 创建服务消费者。

   本示例中，将创建一个服务消费者，通过HSFConsumer所提供的API接口去调用服务提供者。

   i. 创建一个*Maven*工程，命名为*hsf-pandora-boot-consumer*。

   ii. 在*pom.xml*中引入需要的依赖内容。

   > **说明** 消费者和提供者的Maven依赖相同。

```
<properties>
    <java.version>1.8</java.version>
    <spring-boot.version>2.1.6.RELEASE</spring-boot.version>
    <pandora-boot.version>2019-06-stable</pandora-boot.version>
</properties>
<dependencies>
    <dependency>
        <groupId>com.alibaba.boot</groupId>
        <artifactId>pandora-hsf-spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
```

```
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>com.taobao.pandora</groupId>
                <artifactId>pandora-boot-starter-bom</artifactId>
                <version>${pandora-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>com.taobao.pandora</groupId>
                <artifactId>pandora-boot-maven-plugin</artifactId>
                <version>2.1.11.8</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

iii. 将服务提供者所发布的API服务接口（包括包名）拷贝到本地，如com.alibaba.edas.HelloService。

```
public interface HelloService {
    String echo(String string);
}
```

iv. 通过注解的方式将服务消费者的实例注入到Spring的Context中。

```
@Configuration
 public class HsfConfig {
     @HSFConsumer(clientTimeout = 3000, serviceVersion = "1.0.0")
     private HelloService helloService;
 }
```

> **说明** 在 `HsfConfig` 类里配置一次 `@HSFConsumer`，然后在多处通过 `@Autowired` 注入使用。通常一个 `@HSFConsumer` 需要在多个地方使用，但并不需要在每次使用的地方都用 `@HSFConsumer` 来标记。只需要写一个统一的 `HsfConfig` 类，然后在其它需要使用的地方，直接通过 `@Autowired` 注入即可。

v. 为了便于测试，使用SimpleController来暴露一个*/hsf-echo/*的HTTP接口，*/hsf-echo/*接口内部实现调用了HSF服务提供者。

```
@RestController
public class SimpleController {
    @Autowired
    private HelloService helloService;
    @RequestMapping(value = "/hsf-echo/{str}", method = RequestMethod.GET)
    public String echo(@PathVariable String str) {
        return helloService.echo(str);
    }
}
```

vi. 在*resources*目录下的*application.properties*文件中配置应用名与监听端口号。

```
spring.application.name=hsf-pandora-boot-consumer
server.port=8080
spring.hsf.version=1.0.0
spring.hsf.timeout=1000
```

> **说明** 建议将服务版本和服务超时都统一配置在*application.properties*中。

vii. 添加服务启动的main函数入口。

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

服务消费者属性列表

属性	是否必配	描述	类型	默认值
serviceGroup	否	服务的组名	String	HSF

属性	是否必配	描述	类型	默认值
serviceVersion	否	服务的版本号	String	1.0.0.DAILY
clientTimeout	否	客户端统一设置接口中所有方法的超时时间（单位ms）	int	-1
generic	否	是否支持泛化调用	boolean	false
addressWaitTime	否	同步等待服务注册中心（ConfigServer）推送服务提供者地址的时间（单位ms）	int	3000
proxyStyle	否	代理方式（JDK或Javassist）	String	jdk
futureMethods	否	设置调用此服务时需要采用异步调用的方法名列表以及异步调用的方式，默认为空，即所有方法都采用同步调用	String[]	空
consistent	否	负载均衡是否使用一致性哈希	String	空
methodSpecials	否	配置方法级的超时时间、重试次数、方法名称	com.alibaba.boot.hsf.annotation.HSFConsumer.ConsumerMethodSpecial[]	空

服务提供者和消费者全局配置参数列表

属性	是否必配	描述	类型	默认值

属性	是否必配	描述	类型	默认值
spring.hsf.version	否	服务的全局版本号，还可以使用 `spring.hsf.versions.<完整的服务接口名>=<单独为该服务接口设置的版本号，String类型>` ，例如 `spring.hsf.versions.com.aliware.edas.EchoService="1.0.0"` 为具体某个服务设置版本号。	String	1.0.0.DAILY
spring.hsf.group	否	服务的全局组名，还可以使用 `spring.hsf.groups.<完整的服务接口名>=<单独为该服务接口设置的组名，String类型>` 为具体某个服务设置组名。	String	HSF
spring.hsf.timeout	否	服务的全局超时时间，还可以使用 `spring.hsf.timeouts.<完整的服务接口名>=<超时时间，String类型>` 为具体某个服务设置超时时间。	Integer	无

属性	是否必配	描述	类型	默认值
spring.hsf.max-wait-address-time	否	同步等待服务注册中心（ConfigServer）推送服务提供者地址的全局时间（单位ms），还可以使用 `spring.hsf.max-wait-address-times.<完整的服务接口名>=<等待时间，String类型>` 为具体某个服务设置的等待服务注册中心（ConfigServer）推送服务提供者地址的时间。	Integer	3000
spring.hsf.delay-publish	否	服务延迟发布的全局开关，”true”or“false”，还可以使用 `spring.hsf.delay-publishes.<完整的服务接口名>=<是否延迟发布，String类型>` 为具体某个服务设置是否延迟。	String	无
spring.hsf.core-pool-size	否	服务的全局最小活跃线程数，还可以使用 `spring.hsf.core-pool-sizes.<完整的服务接口名>=<最小活跃线程数，String类型>` 单独为某服务设置最小活跃线程数。	int	无

属性	是否必配	描述	类型	默认值
spring.hsf.max-pool-size	否	服务的全局最大活跃线程数，还可以使用 `spring.hsf.max-pool-sizes.<完整的服务接口名>=<最大活跃线程数，String类型>` 单独为某服务设置最大活跃线程数。	int	无
spring.hsf.serialize-type	否	服务的全局序列化类型，Hessian或者Java，还可以使用 `spring.hsf.serialize-types.<完整的服务接口名>=<序列化类型>` 单独为某服务设置序列化类型。	String	无

> **说明** 全局配置参数可在Pandora Boot应用的*application.properties*文件中设置。

本地开发调试

1. 配置轻量级配置及注册中心。

   本地开发调试时，需要使用轻量级配置及注册中心，轻量级配置及注册中心包含了服务注册发现服务端的轻量版，详情请参见启动轻量级配置及注册中心。

2. 启动应用。

   - 在IDE中启动
     通过VM options配置启动参数*-Djmenv.tbsite.net={$IP}*，通过main方法直接启动。其中 `{$IP}` 为轻量配置中心的IP地址。例如本机启动轻量配置中心，则 `{$IP}` 为 `127.0.0.1` 。
     您也可以不配置JVM的参数，而是直接通过修改hosts文件将 `jmenv.tbsite.net` 绑定为轻量配置中心的IP。详情请参见启动轻量级配置及注册中心。
   - 通过FatJar启动
     a. 增加taobao-hsf.sar依赖，这样会下载到我们需要的依赖：/.m2/com/taobao/pandora/taobao-hsf.sar/2019-06-stable/taobao-hsf.sar-2019-06-stable.jar，在后面的启动参数中依赖它。

```
<dependency>
     <groupId>com.taobao.pandora</groupId>
     <artifactId>taobao-hsf.sar</artifactId>
     <version>2019-06-stable</version>
 </dependency>
```

b. 使用Maven将Pandora Boot工程打包成FatJar， 需要在*pom.xml*中添加如下插件。为避免与其他打包插件发生冲突，请勿在build的plugin中添加其他FatJar插件。

```
<plugin>
    <groupId>com.taobao.pandora</groupId>
    <artifactId>pandora-boot-maven-plugin</artifactId>
    <version>2.1.11.8</version>
    <executions>
        <execution>
        <phase>package</phase>
        <goals>
            <goal>repackage</goal>
        </goals>
        </execution>
    </executions>
</plugin>
```

c. 添加完插件后，在工程的主目录下，执行Maven命令 `mvn clean package` 进行打包，即可在*Target*目录下找到打包好的FatJar文件。

d. 通过Java命令启动应用。

```
java -Djmenv.tbsite.net=127.0.0.1 -Dpandora.location=${M2_HOME}/.m2/repository/com/taobao/pandora/taobao-hsf.sar/2019-06-stable/taobao-hsf.sar-2019-06-stable.jar -jar hsf-pandora-boot-provider-1.0.jar
```

> **说明** `-Dpandora.location` 指定的路径必须是全路径，使用命令行启动时，必须显示指定taobao-hsf.sar的位置。

访问consumer所在机器的地址，可以触发consumer远程调用provider。

```
curl localhost:8080/hsf-echo/helloworld
helloworld
```

单元测试

Pandora Boot的单元测试可以通过PandoraBootRunner启动，并与SpringJUnit4ClassRunner无缝集成。

我们将演示一下如何在服务提供者中进行单元测试，供大家参考。

1. 在Maven中添加Pandora Boot和Spring Boot测试必要的依赖。

```
<dependency>
    <groupId>com.taobao.pandora</groupId>
    <artifactId>pandora-boot-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2. 编写测试类的代码。

```
@RunWith(PandoraBootRunner.class)
  @DelegateTo(SpringJUnit4ClassRunner.class)
  // 加载测试需要的类，一定要加入Spring Boot的启动类，其次需要加入本类。
  @SpringBootTest(classes = {HSFProviderApplication.class, HelloServiceTest.class })
  @Component
  public class HelloServiceTest {
      /**
       * 当使用 @HSFConsumer时，一定要在 @SpringBootTest类加载中，加载本类，通过本类来注入对象，否则当做泛化时，会出现类转换异常。
       */
      @HSFConsumer(generic = true)
      HelloService helloService;
      //普通的调用。
      @Test
      public void testInvoke() {
          TestCase.assertEquals("hello world", helloService.echo("hello world"));
      }
      //泛化调用。
      @Test
      public void testGenericInvoke() {
          GenericService service = (GenericService) helloService;
          Object result = service.$invoke("echo", new String[] {"java.lang.String"}, new Object[] {"hello world"});
          TestCase.assertEquals("hello world", result);
      }
      //返回值Mock。
      @Test
      public void testMock() {
          HelloService mock = Mockito.mock(HelloService.class, AdditionalAnswers.delegatesTo(helloService));
          Mockito.when(mock.echo("")).thenReturn("beta");
          TestCase.assertEquals("beta", mock.echo(""));
      }
  }
```