# 一：概述

## 1.背景介绍

ServiceComb 作为 Apache 开源组织下的一款微服务框架，其前身为华为云的 微服务引擎 CSE (Cloud Service Engine) 云服务。它意味着国内一款微服务框架在华为和 Apache 组织的共同努力下，随着微服务市场的火爆，一定会让越来越多的开发者所喜欢。

## 2.首要原则

全球首款进入 Apache 的开源微服务项目，中立、开放、标准、无商业 Lock-in
开源与商业代码同源，具备零成本平滑迁移商用的能力，社区长足发展有保障

## 3.技术方案

解决方案级，多语言、多通信协议、标准服务契约、事务最终一致性
开源开放，拥抱 SpringBoot、SpringCloud、ServiceMesh 等主流生态
低门槛准入，业务侵入度低，架构松耦合

## 4.官方网站介绍

华为将 ServiceComb 贡献给了 Apache 基金组织后，我们就可以通过 Apache 的官方网站提供的资料来学习 ServiceComb，下面是官网地址：

英文：http://servicecomb.incubator.apache.org/
中文：http://servicecomb.incubator.apache.org/cn/

# 二、 ServiceComb 与 SpringCloud 的比较

我们可以从语言框架，编程模型，通信协议，服务治理，服务访问，分布式事务等方面进行比较，得出结果如下：

| 领域 | 功能 | SpringCloud | ServiceComb |
|---|---|---|---|
| 多语言框架 | JAVA 微服务框架 | √ | √ |
| | 非侵入微服务框架 | × | √ |
| 多编程模型 | POJO 编程模型 | × | √ |
| | SpringMVC 编程模型 | √ | √ |
| | JAX-RS 编程模型 | × | √ |
| | Reactive 响应式编程 | × | √ |
| 多通信协议 | Rest | √ | √ |
| | RPC | × | √ |

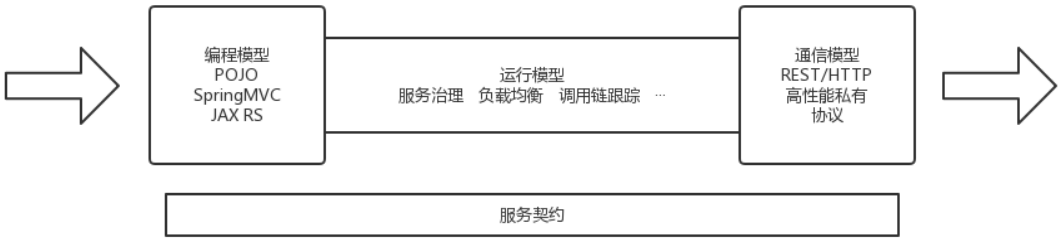
| | | | |
|---|---|---|---|
| 服务治理 | 负载均衡 | √ | √ |
| | 服务限流 | √ | √ |
| | 服务隔离 | √ | √ |
| | 服务容错 | √ | √ |
| | 服务熔断 | √ | √ |
| | 错误注入 | × | √ |
| | 服务中心 | √ | √ |
| | 配置中心 | √ | √ |
| 服务访问 | 安全协议 | √ | √ |
| | 安全认证 | √ | √ |
| | 服务契约 | √ | √ |
| 分布式事务 | 事务一致性方案 | × | √ |

通过上图的对比，我们可以得出结论就是 ServiceComb 在微服务开发上更胜一筹。我们有理由相信 ServiceComb 将会在微服务开发领域成为国人的骄傲。

# Java Chassis系统架构

开放性设计思想

## 框架概述




## 主要设计意图

1.编程模型和通信模型分离，不同的编程模型可以灵活组合不同的通信模型。应用开发者在开发阶段只关注接口开发，部署阶段灵活切换通信方式；支持legacy系统的切换，legacy系统只需要修改服务发布的配置文件（或者annotation），而不需要修改代码。

现阶段支持SpringMVC、JAX-RS和透明RPC三种开发方式。

2.内建API-first支持。通过契约规范化微服务开发，实现跨语言的通信，并支持配套的软件工具链（契约生成代码、代码生成契约等）开发，构建完整的开发生态。

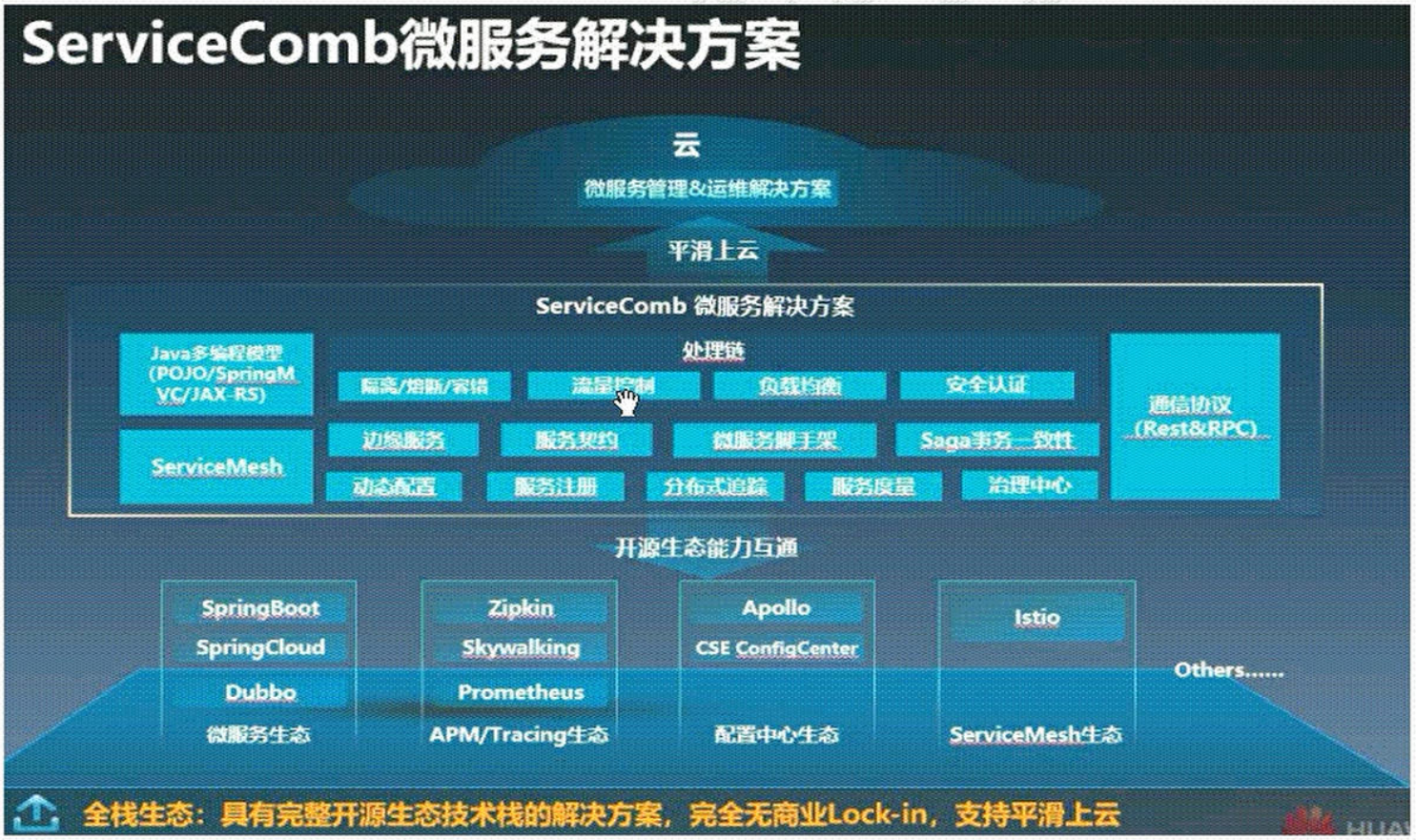
3.定义了常用的微服务运行模型，将微服务从发现到交互过程中的各种容错手段都封装起来。该运行模型支持自定义和扩展。

## 模块说明

| 类型 | artifact id | 是否可选 | 功能说明 |
|---|---|---|---|
| 编程模型 | provider-pojo | 是 | 提供RPC开发模式 |
| 编程模型 | provider-jaxrs | 是 | 提供JAX RS开发模式 |
| 编程模型 | provider-springmvc | 是 | 提供Spring MVC开发模式 |
| 通信模型 | transport-rest-vertx | 是 | 运行于HTTP之上的开发框架，不依赖WEB容器，应用打包为可执行jar |
| 通信模型 | transport-rest-servlet | 是 | 运行于WEB容器的开发框架，应用打包为war包 |
| 通信模型 | transport-highway | 是 | 提供高性能的私有通信协议，仅用于java之间互通。 |
| 运行模型 | handler-loadbalance | 是 | 负载均衡模块。提供各种路由策略和配置方法。一般客户端使用。 |
| 运行模型 | handler-bizkeeper | 是 | 和服务治理相关的功能，比如隔离、熔断、容错。 |
| 运行模型 | handler-tracing | 是 | 调用链跟踪模块，对接监控系统，吐出打点数据。 |

四：ServiceComb微服务解决方案

设计理念底层模块分析




## 五、 安装 ServiceComb 开发环境

应用开发环境所需安装的工具包括 JDK、Maven、Eclipse 和 IDEA 。

现在我们介绍如何在开发者本地进行消费者/提供者应用的开发调试。开发服务提供者和消费提供者均需要连接到在远程的服务中心，为了本地微服务的开发和调试：

启动本地服务中心；

服务中心是微服务框架中的重要组件，用于服务元数据以及服务实例元数据的管理和处理注册、发现。服务中心与微服务提供/消费者的逻辑关系下图所示：

六：服务注册中心CSE介绍及原理分析

JDK8

Maven 3.5.2

IDEA

服务注册中心CSE：Cloud Service Engine 云服务引擎

好处：可以轻松的将微服务发布到云平台 华为云PaaS平台

用ServiceComb开发微服务，再借用华为云PaaS平台实现微服务治理

服务提供者

提供微服务功能

服务注册中心

注册微服务，实现微服务的管理

服务消费者

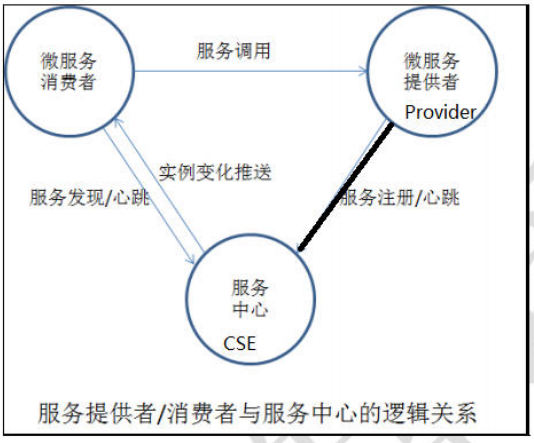
调用微服务

1.服务提供者向CSE注册中心注册一个微服务

2.服务消费者就可以向CSE注册中心服务发现，

3.如果在CSE注册中心找到了要调用的微服务就可以发送一个微服务的实例给服务消费者。

4.心跳：通过心跳的消息实现微服务状态检测




服务提供者/消费者与服务中心的逻辑关系

2．服务注册中心CSE安装和目录结构

直接解压

apache-servicecomb-incubating-service-center-1.0.0-m2-windows-amd64.tar.gz　2019/4/16 上午 10:02　WinRAR 压缩文件


打开 conf/app.conf 文件后，可以找到 CSE 基本配置如下：

服务端配置：

```
# sever options
##############################################################################
# if you want to listen at ipv6 address, then set the httpaddr value like:
# httpaddr = 2400:A480:AAAA:200::159        (global scope)
# httpaddr = fe80::f816:3eff:fe17:c38b%eth0 (link-local scope)
httpaddr = 127.0.0.1 ip,远程需要改IP地址
httpport = 30100   微服务注册的端口

read_header_timeout = 60s
read_timeout = 60s
idle_timeout = 60s
write_timeout = 60s
max_header_bytes = 32768 # 32K
max_body_bytes = 2097152 # 2M

enable_pprof = 0
```

前端配置：

```
# Frontend Configurations
##############################################################################
frontend_host_ip = 127.0.0.1
frontend_host_port = 30103
```

2 .启动本地服务中心后，在服务提供/消费者的 microservice.yaml 文件中配置 ServerCenter 的地址和端口，示例代码：

```
servicecomb:
 service:
 registry:
 address:
http://127.0.0.1:30100
 #服务中心地址及端口
```

## 七、 使用官方提供的脚手架快速开发 ServiceComb

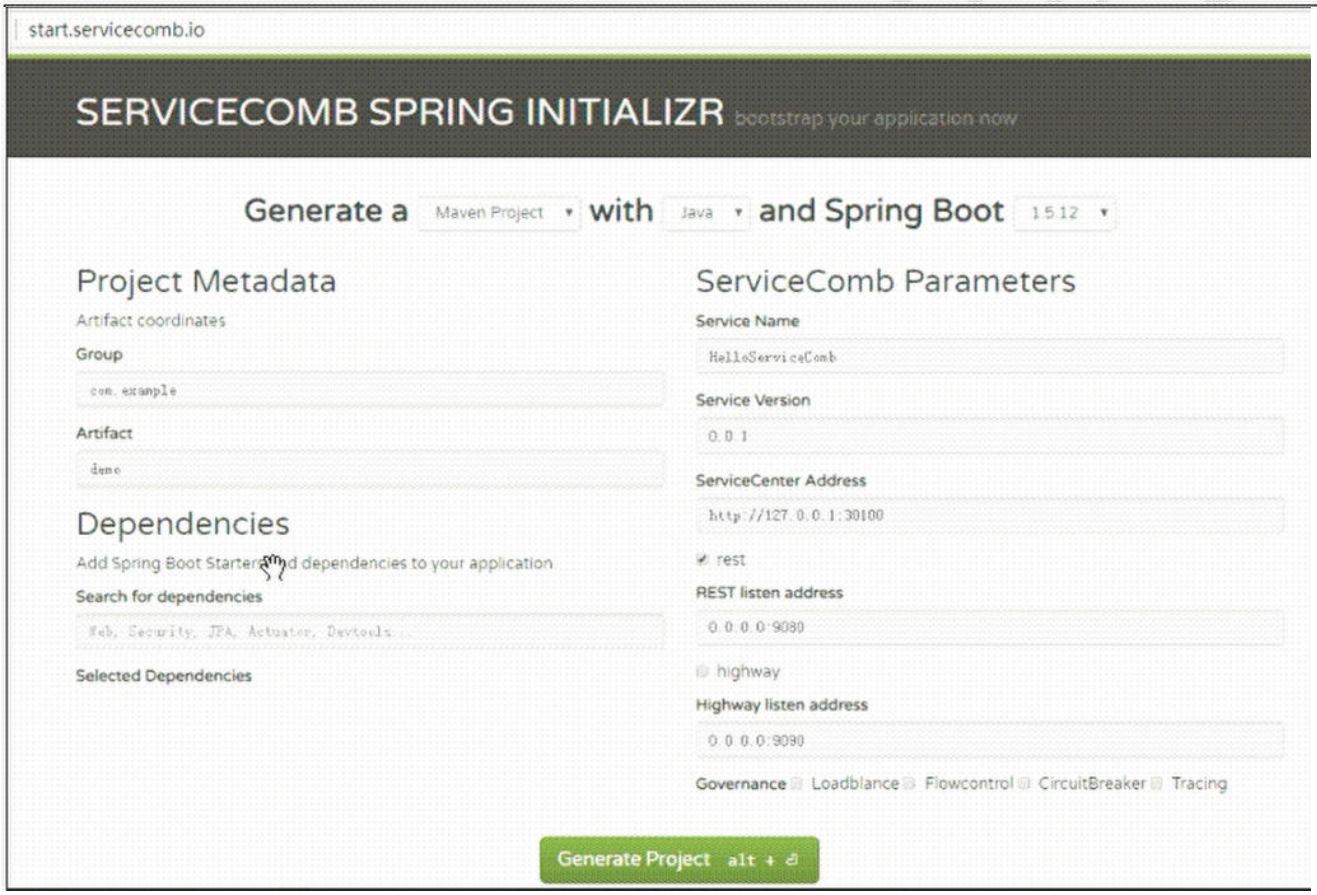
无法打开

为了能够使开发者可以快速构建 ServiceComb 应用程序，它同样也为我们提供了一套脚手架，这样能够方便学习者及应用开发者快速入门，同时极大的提高了效率。

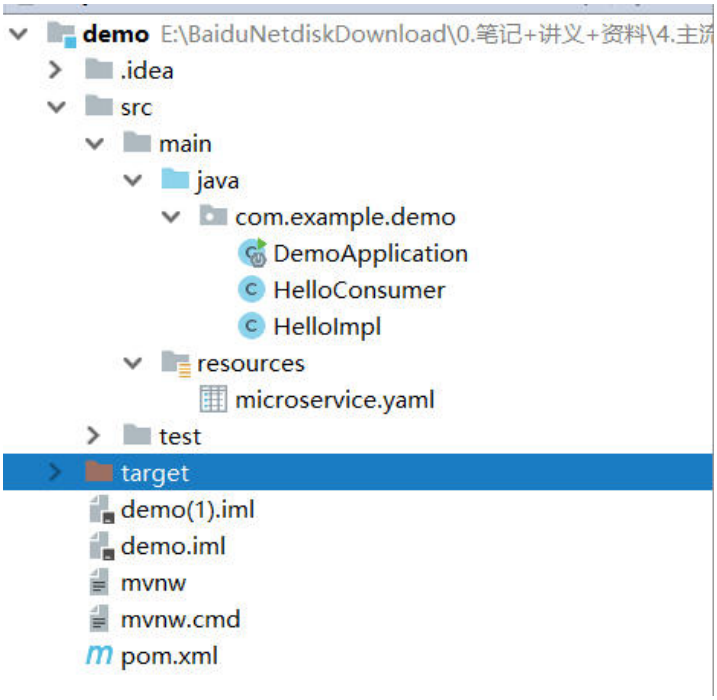
1.访问快速开发引导页：

http://start.servicecomb.io/ http://start.servicecomb.io/

页面如下：




下载一个入门级的demo



ServiceComb 入门程序的分析

pom.xml文件

继承一个父工程,将面向对象的继承思想进一步放大。

Maven中的父子工程的继承，可以将面向对象中的继承思想进一步发扬。

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.12.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

点开它的源码，进入spring-boot-starter-parent这个pom文件

打开：

```
  <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-dependencies</artifactId>
  <version>1.5.12.RELEASE</version>
  <relativePath>../../spring-boot-dependencies</relativePath>
  </parent>
```

***面向对象的继承是单一的***

***继承一个父类，同时再去实现一个接口***

***如果一个pom文件，同时要继承两个父的pom文件***

***1.parent***

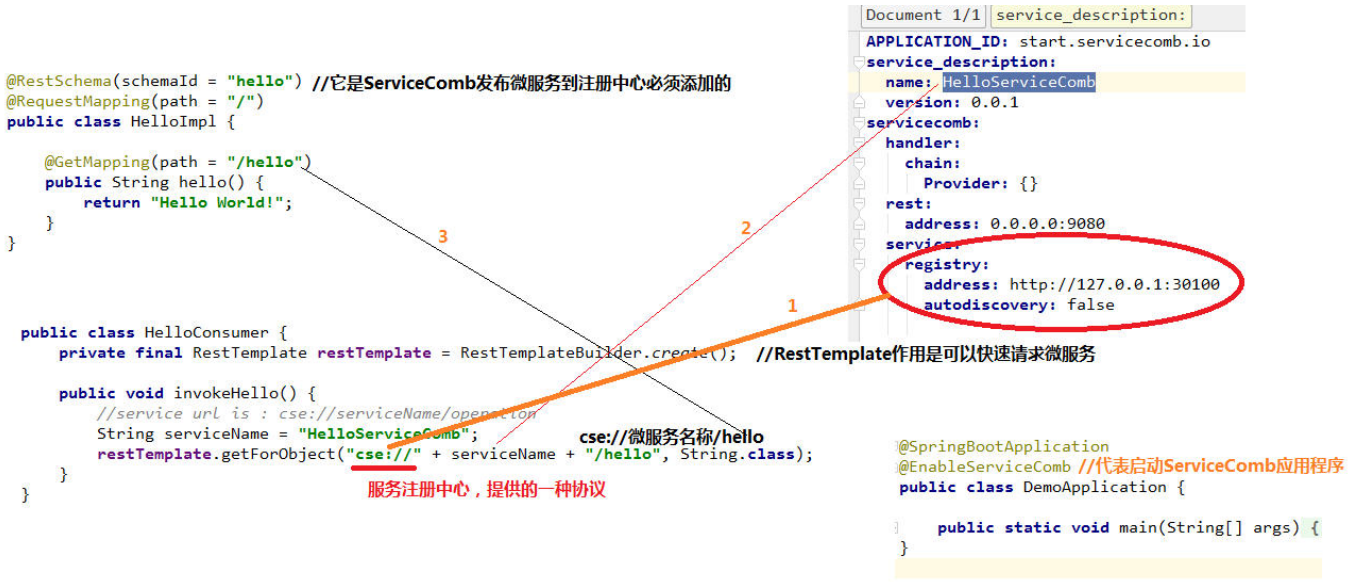
***2.dependencyManagement*** 依赖管理

```
<type>pom</type>
<scope>import</scope>
```

**microservice.yaml文件**

```
APPLICATION_ID: start.servicecomb.io  #应用程序编号
service_description:
  name: HelloServiceComb #应用程序的名称
  version: 0.0.1
servicecomb:
  handler:
    chain:
      Provider: {}
  rest: #rest通信
    address: 0.0.0.0:9080 #后续会用到这个端口
  service:
    registry:
      address: http://127.0.0.1:30100  #服务注册中心CSE的地址
      autodiscovery: false
```

## 入门程序运行分析




**总结**

**1.pom.xml文件中加入springboot,servicecomb依赖**

**多继承问题的解决**

**2.microservice.yaml文件**

**微服务的名称，通信协议，服务注册中心的信息**

**3.开发服务提供者**

**其实本身就是一个SpringMVC的Controller**

`@RestSchema(schemaId = "hello")`放在类上

4.服务消费者

RestTemplate

getForObject(URL);

URL: cse://微服务名称/微服务方法

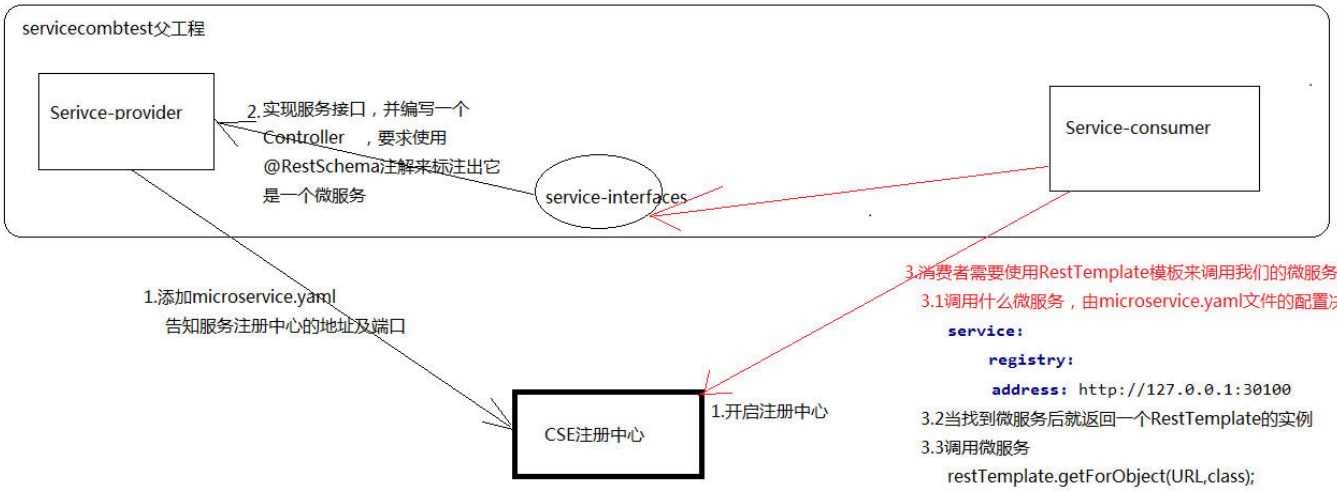

5.访问微服务的提供者发布的方法

cse注册中心的IP

通信协议：端口

方法：

## 八：Rest编程开发---框架程序 面向资源位置




分析程序，看一下配置文件是怎么写的

5.测试效果 消费者的microservice.yaml rest http://localhost:9081/test/test?name=you

开

## 九：Service的线程通信模型介绍

微服务（分布式开发）

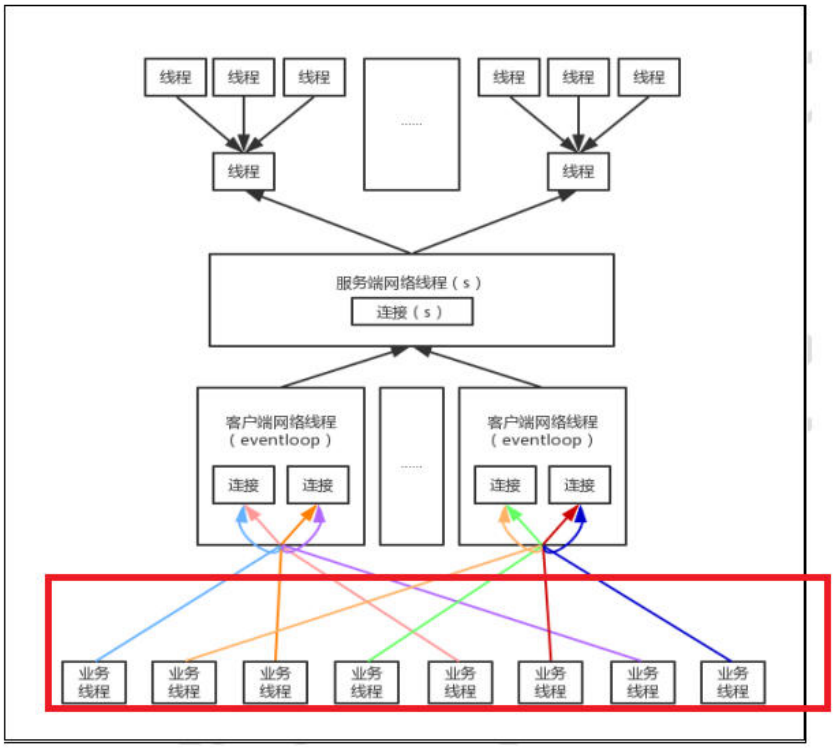
技术方案：

1.dubbo+zookeeper

2.SpringCloud+Eureka

3.ServiceComb+CSE

通信模型（Vertx,HighWay,REST）




## 十：RPC编程开发---框架程序 面向方法调用 看程序

RPC方式开发ServiceComb应用程序

步骤：

1.框架结构：一个父工程，三个子模块(服务接口，服务提供者，服务消费者)

2.开发服务接口，安装服务接口到本地仓库 需要先把maven插件注释掉

3.开发服务提供者,导入坐标

```
<!--rpc通信模型-->
<dependency>
<groupId>org.apache.servicecomb</groupId>
<artifactId>transport-highway</artifactId>
</dependency>
<!--rpc编程模型-->
<dependency>
<groupId>org.apache.servicecomb</groupId>
<artifactId>provider-pojo</artifactId>
</dependency>
```

4.开发服务提供者的类

@RpcSchema(schemaId="")；类上面添加的注解

5.开发服务提供者的启动类

6.配置microservice.yaml文件

```
highway:
  address: 0.0.0.0:9090
```

7.开发服务消费者

7.1引入pom文件

要引入transport-highway,provider-pojo这两个坐标

7.2开发服务消费者的实现类

@RpcReferences(microserviceName="",schemaId="")

7.3开发Controller

@AutoWire来注入消费者的实现类

7.4开发启动类

7.5开发microservice.yaml文件

```
highway:
    address: 0.0.0.0:9091
```

7.6指定应用服务器的端口

application.properties

server.port

# 十一、ServiceComb 服务治理方案

为了引入 ServiceComb 的服务治理策略，我们可以加入相关配置。
首先，需要在 pom.xml 文件中加入相关的坐标

```xml
<!--限流-->
<dependency>
    <groupId>org.apache.servicecomb</groupId>
    <artifactId>handler-flowcontrol-qps</artifactId>
</dependency>
<!--熔断包-->
<dependency>
    <groupId>org.apache.servicecomb</groupId>
    <artifactId>handler-bizkeeper</artifactId>
</dependency>
<!--日志追踪-->
<dependency>
    <groupId>org.apache.servicecomb</groupId>
    <artifactId>handler-tracing-zipkin</artifactId>
</dependency>
```

在microservice文件中加入如下配置

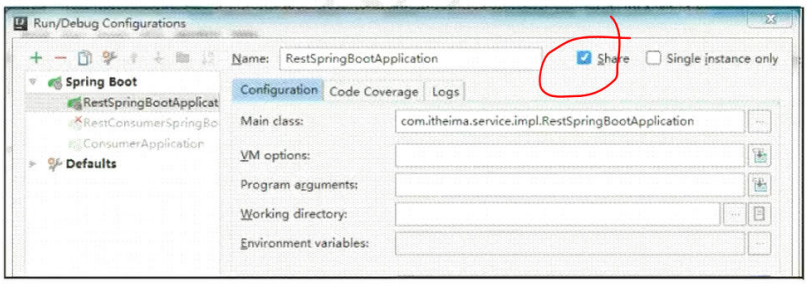
```yaml
servicecomb:
  circuitBreaker:
    Provider:
      provider:
        requestVolumeThreshold: 8
  fallbackpolicy:
    provider:
      policy: returnnull
  flowcontrol:
    Provider:
      qps:
        limit:
          gateway: 1
  handler:
    chain:
      Provider:
        default: qps-flowcontrol-provider,bizkeeper-provider
```

ServiceComb 提供了基于 Ribbon 的负载均衡方案，用户可以通过配置文件配置负载均衡策略，当前支持随机、顺序、基于响应时间的权值等多种负载均衡路由策略。

## 1.负载均衡策略

启动多个service实例

作为 ServiceComb 内置策略，我们测试一下执行效果。
首先，将服务提供者的启动类设置为共享（Share）模式。如下图所示：




其次：修改服务提供者的 Service 类，加入一行输出代码。

```java
@RestSchema(schemaId = "hello")
@RequestMapping("/hello")
public class RestServiceImpl implements RestService {
    @Override
    @GetMapping("/hello")
    public String sayRest(String name) {
        System.out.println("服务提供者-1");
        return "Hello World "+name;
    }
```

## 2.限流策略

限流是微服务框架基本都可以解决的一个策略，是微服务框架中常见的系统保障措施。通常来说系统的吞吐量是可以提前预测的，当请求量超过预期的伐值时可以采取一些限制措施来保障系统的稳定运行，比如延迟处理、拒绝服务等。

ServiceComb 微服务框架限流主要是基于 zuul 网关来实现限流的。通常需要先配置好 zuul 网关。

本次实验：我们再次添加一个 gate 网关模块。

第一步：添加 pom.xml 文件中的 Zuul 网关依赖：

```
<!-- zuul proxy 需要的包-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zuul</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
<dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
 <dependency>
     <groupId>org.apache.servicecomb</groupId>
      <artifactId>spring-cloud-zuul-zipkin</artifactId>
 </dependency>
```

第二步：添加 application.yml 文件,配置如下：

```
server:
  port: 9003
zuul:
  routes:
    shicifang-friend:
      serviceId: shicifang-friend
    shicifang-qa:
      serviceId: shicifang-qa
discoveryServer:
```

```
  ribbon:
    eureka:
      enabled: false
servicecomb:
  tracing:
    enabled: true
```

第三步：添加 Zuul 网关的启动类

```
@SpringBootApplication
@EnableServiceComb
@EnableZuulProxy//新增注
public class ZuulSpringBootApplication {
    public static void main(String[] args) {
        SpringApplication.run(ZuulSpringBootApplication.class,args);
    }
    @Bean
    public CorsFilter corsFilter() {
        final UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        final CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true); // 允许 cookies 跨域
        config.addAllowedOrigin("*");// #允许向该服务器提交请求的 URI，*表示全部允许，在
SpringMVC 中，如果设成*，会自动转成当前请求头中的 Origin
        config.addAllowedHeader("*");// #允许访问的头信息,*表示全部
        config.setMaxAge(18000L);// 预检请求的缓存时间（秒），即在这个时间段里，对于相同的跨域请
求不会再预检了
        config.addAllowedMethod("OPTIONS");// 允许提交请求的方法，*表示全部允许
        config.addAllowedMethod("HEAD");
        config.addAllowedMethod("GET");// 允许 Get 的请求方法
        config.addAllowedMethod("PUT");
        config.addAllowedMethod("POST");
        config.addAllowedMethod("DELETE");
        config.addAllowedMethod("PATCH");
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
```

第四步：在 ServiceComb 的配置文件 mircoservice.yaml 中添加如下配置：

```
servicecomb:
 flowcontrol:
  Provider:
   qps:
     limit:
       gateway: 1000
 handler:
  chain:
   Provider:
     default: qps-flowcontrol-provider,bizkeeper-provider,tracing-provider
```

WEBUI 测试频繁刷新 10 次观看查询次数查看调用次数，有部分服务调用没有成功

System.out.println("微服务 1 调用");

把 1 修改为 1000， 频繁访问 WEBUI 测试地址。查看调用次数

System.out.println("微服务 1 调用"); http:状态码 429 太多访问