腾讯云

TSF 腾讯分布式服务

中间件产品解决方案

腾讯云

中间件产品解决方案 —— 全面覆盖微服务治理、系统异步解耦、负载均衡、加密管理等技术难题

业界优势

腾讯云运营着全球**规模最大**的

KVM虚拟化云服务器集群，提供最专业的公有云、混合云服务

➢业内性能最强**TencentGateway-负载均衡服务**，承载腾讯集团如微信、手机QQ、王者荣耀等核心业务，峰值流量超过10TB

➢ 腾讯云分布式服务治理框架：基于**Spring Cloud开源方案**，提供一站式的服务注册发现，服务调用。代码发布、回滚，调用链跟踪等整体微服务解决方案

➢ 中间件服务：提供企业级**消息队列、ESB服务总线、云API网关、KMS秘钥管理，HSM加密机**服务

➢ 推出FAAS产品：**Function as a Service**，函数即服务。无需购买服务器即可快速部署服务

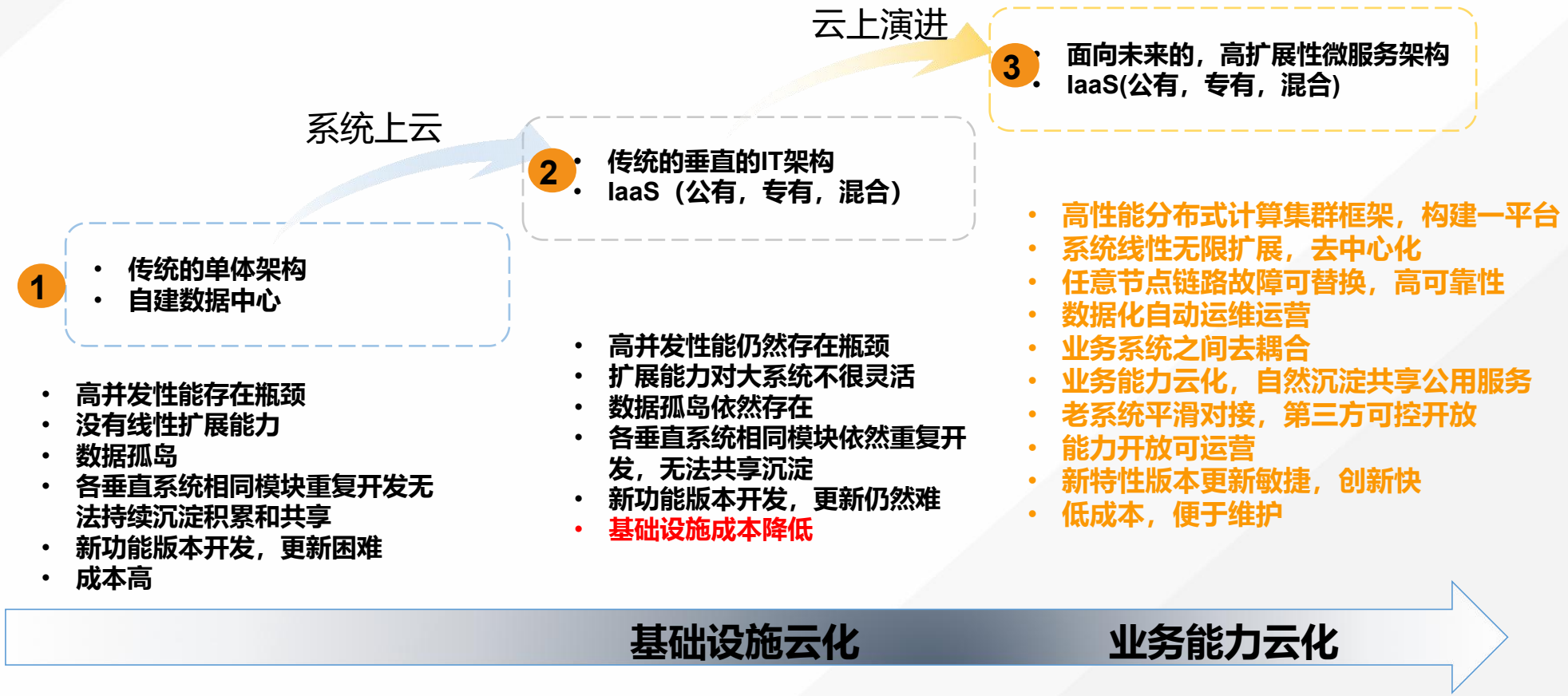
一站式解决方案

- 服务治理解决方案
- 消息队列服务
  ESB消息总线
- 负载均衡、
  API网关
- 加密机、
  秘钥管理服务
- 无服务器
  函数计算
  HOT

腾讯云

一、为什么要微服务化?

企业IT架构的演进



云计算时代，企业信息化演进不仅仅是把IT系统搬到云上，而是让业务与信息系统深度融合，改变业务运营和创新模式。

微服务的发展过程

互联网架构转型

微服务的开发迭代不需要协调其他业务组件，可自主的选择技术框架。可独立自主的进行敏捷业务部署、扩展

单体架构

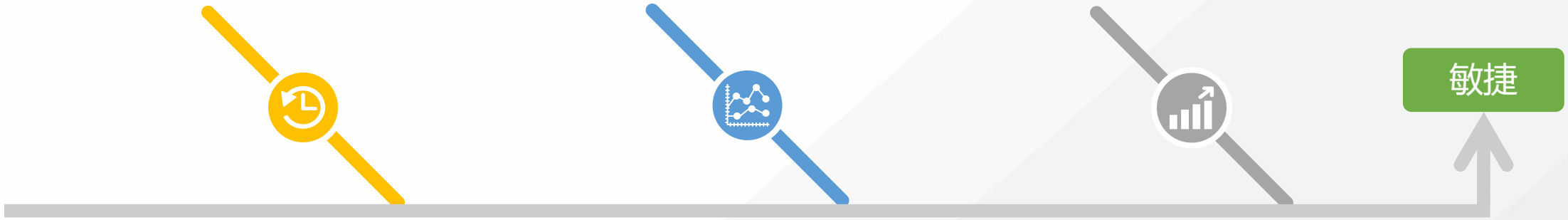
塔式应用、单体应用，扩展困难，维护困难

SOA

大型系统分层、解耦，标准接口调用，分布式系统

Microservices

分布式系统云计算时代产物，关注敏捷交付和部署速度、频次。提供高级别的系统水平扩展能力



微服务的优势和挑战

独立的可伸缩

每个微服务可以在不影响其他微服务的情况下进行功能扩展

复用概率高

每个微服务都可以独立地伸缩
可以更加直观定位性能瓶颈
数据库分片可以根据需求来实施

独立的技术栈

不需要同一标准化技术栈的选择，无需针对技术选型而纠结，关注于业务实现
无需引入未被使用的技术或库
根据业务实现需求来选择最合适的技术

业务功能独立

更新新版本界面，不需要更新整个系统，可以进行整个业务功能的重写，并替换之

代码库独立

每个微服务具备自己的代码仓库，由对应团队开发者维护，编译、打包、发布及部署都很快，服务启动迅速。在各个服务的代码库间没有交叉依赖

微服务架构，存在的问题?

1、服务之间如何发现？

2、如何理清复杂的服务依赖关系？

3、服务健康状况（如访问量、响应时间、并发数等）如何监控？

4、如何对服务进行扩缩容？

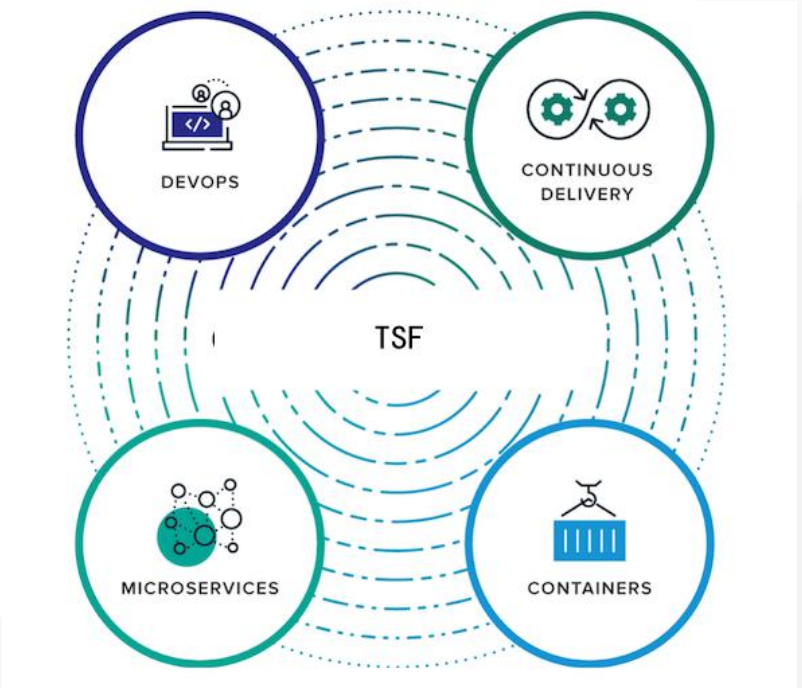
5、如何做容量规划？

6、如何保证服务的安全访问（权限控制）？

腾讯分布式服务框架（Tencent Service Framework）

腾讯云

腾讯云分布式服务治理平台，是一个围绕服务注册发现、管理、微服务、服务持续集成的PaaS平台，提供多样的应用发布能力和轻量级微服务解决方案。



腾讯分布式服务框架（Tencent Service Framework）

腾讯云分布式服务治理平台，是一个围绕服务注册发现、管理、微服务、服务持续集成的PaaS平台，提供多样的应用发布能力和轻量级微服务解决方案。**子产品包括：**

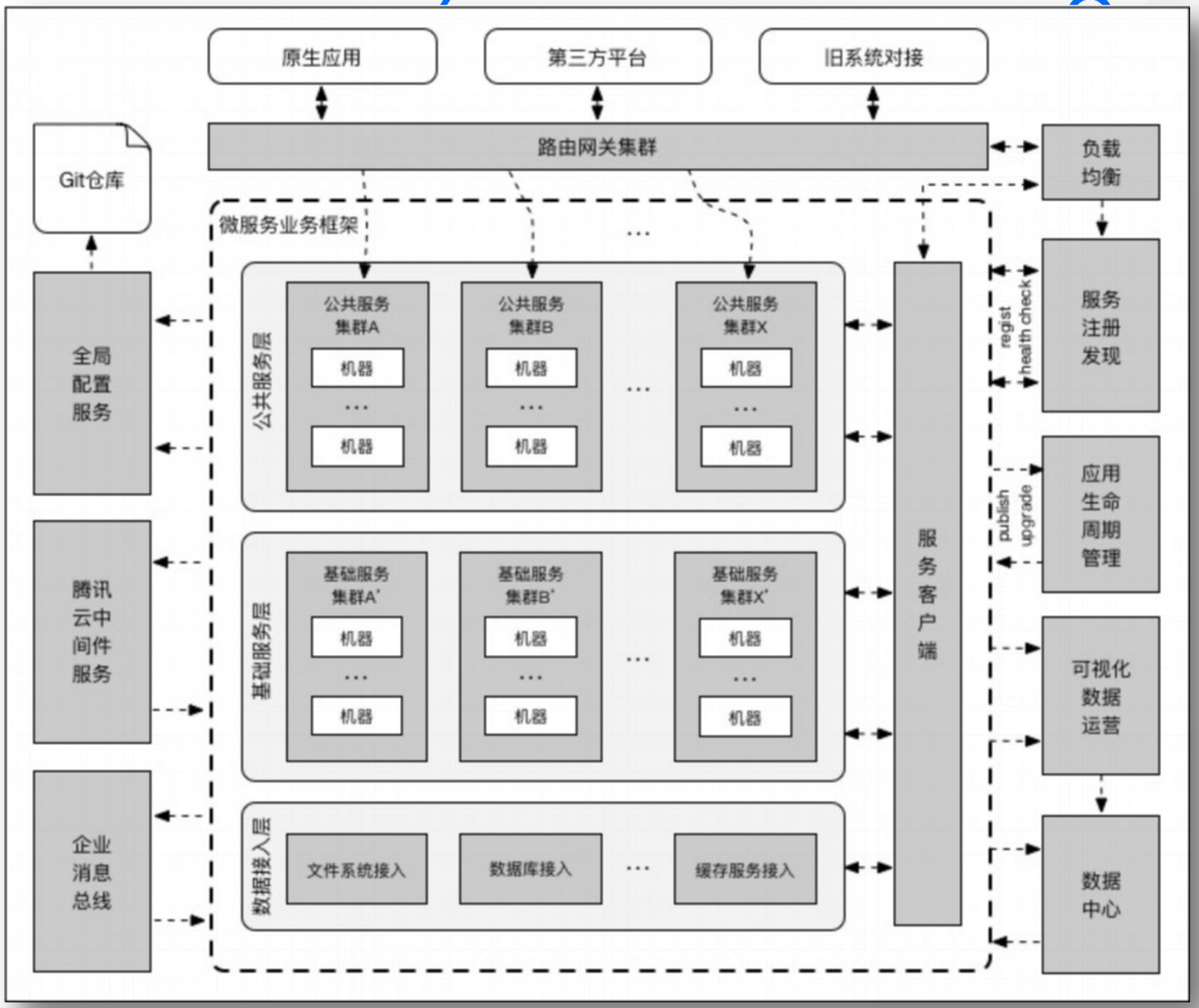
API 网关

消息队列

分布式配置

分布式事务

分布式计算调度等能力



腾讯分布式服务框架（Tencent Service Framework)

TSF基于开源的Spring Cloud框架，提供全套的服务治理、服务生命周期管理，调用链跟踪等服务。Java开发者迁移上云“零门槛”！

星火燎原的Spring Cloud

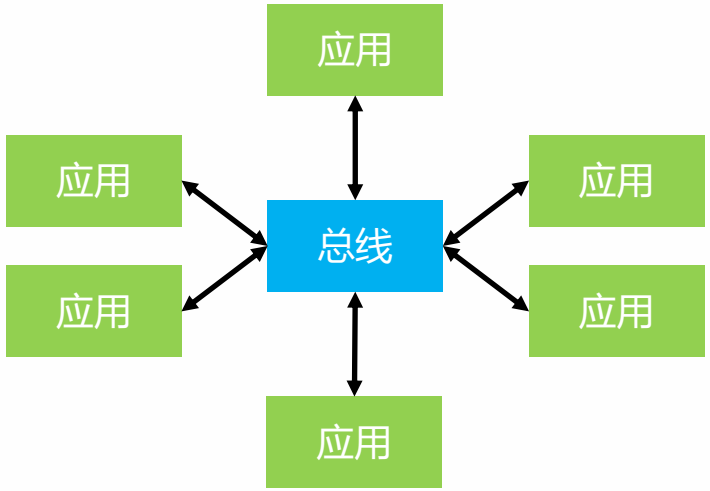
中国联通子公司
上海米么金服
指点无限（北京）科技有限公司
易保软件
广州简法网络
深圳睿云智合科技有限公司
猪八戒网
上海云首科技有限公司
华为
东软
南京云帐房网络科技有限公司
四众互联(北京)网络科技有限公司
深圳摩令技术科技有限公司
广州万表网
视觉中国
上海秦苍信息科技有限公司-买单侠
爱油科技(大连)有限公司
冰鉴科技
拍拍信
有赞
广发证券深圳金融科技研发中心
....
数据来源：http://docs.springcloud.cn/

TSF > spring cloud

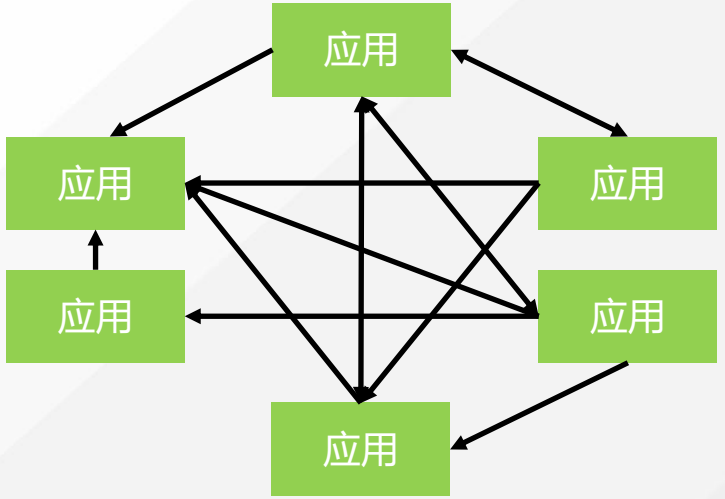
1、支持分布式事务
2、支持分布式动态配置，友好界面
3、服务注册中心支持多租户隔离
4、完整的生命周期管理能力
5、服务调用鉴权
6、支持rpc通信能力
7、springcloud多组件整合(集成到控制台)
8、提供与腾讯云服务深度整合能力
9、提供dubbo到springcloud的集成能力
10、提供完整的调用链解决方案
11、api网关打通，并限速限流以及协议转换的能力
12、提供完整的业务日志展现能力
13、提供docker集成的整体解决方案

腾讯分布式服务框架（Tencent Service Framework）

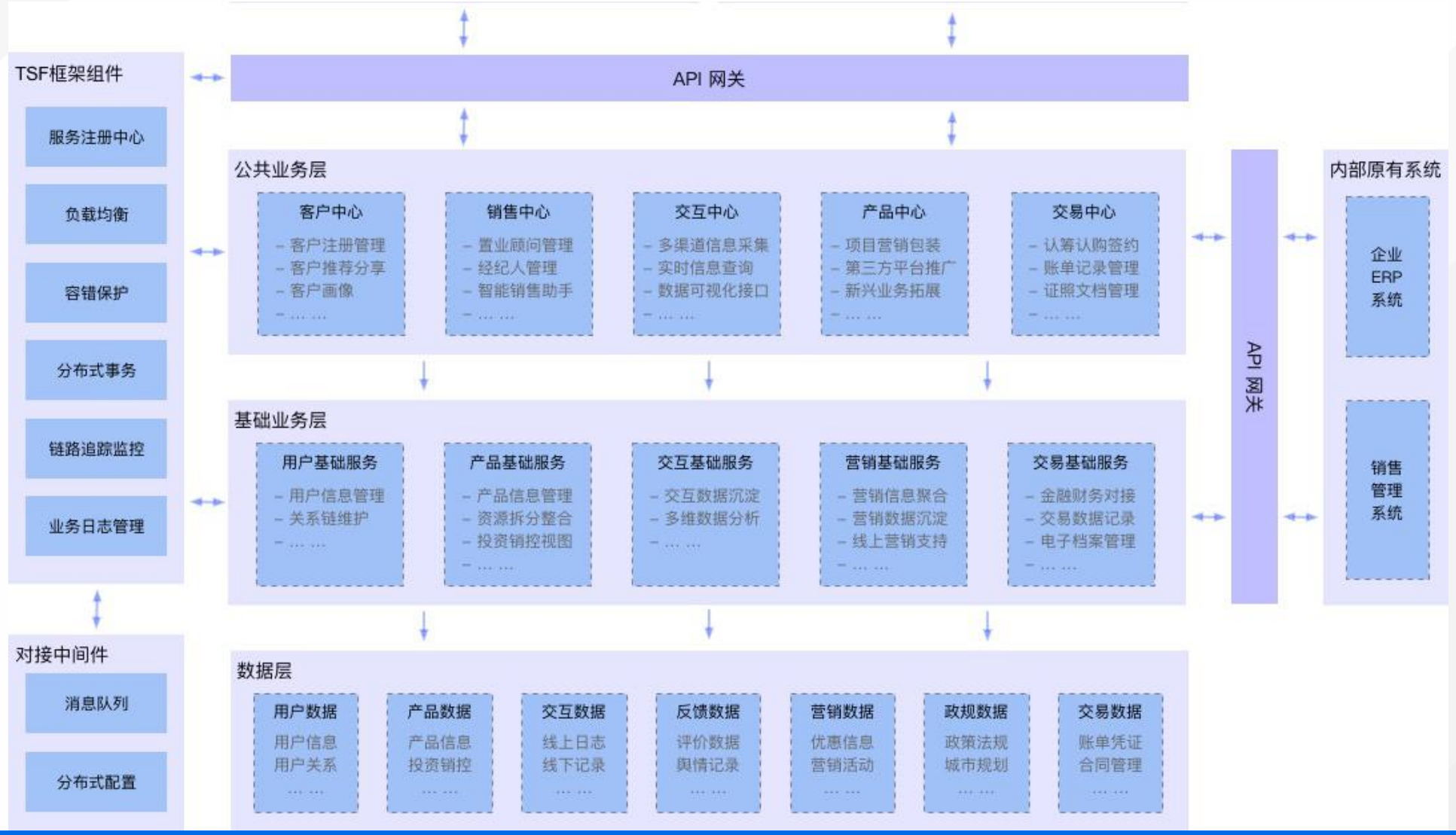
微服务框架核心其一：去中心化服务治理，提供高可靠的、去中心化的服务注册、调用、服务治理平台



中心化 vs 去中心化



腾讯分布式服务框架（Tencent Service Framework）–地产行业微服务拆分



腾讯分布式服务框架（Tencent Service Framework)

当集中式应用转变成分布式系统后，系统之间的相互可靠调用一直以来都是分布式架构的难题，比如网络通信，序列化协议设计等很多技术细节需要确定。

TSF提供了一个高性能的 restful/rpc 调用 框架，能够构建高可用的分布式系统，系统地考虑各个应用之间的分布式服务发现、服务路由、服务调用以及服务安全等细节

腾讯分布式服务框架（Tencent Service Framework）

微服务框架核心其二：服务自动化部署

腾讯分布式服务框架（Tencent Service Framework）

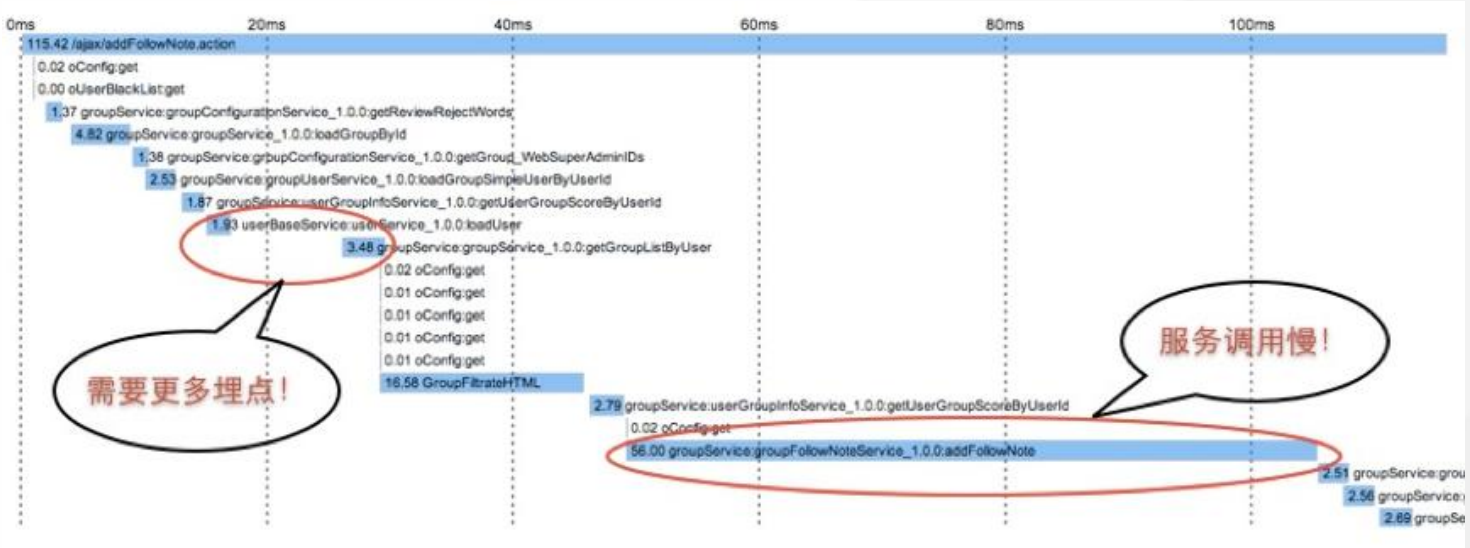
腾讯云

微服务框架核心其三：完善的监控分析

1、分布式调用链跟踪管理，协助快速定位问题，发现系统瓶颈所在

2、成熟的数据化运维/运营工具

3、服务数据可视化



腾讯云

腾讯分布式服务框架（Tencent Service Framework)

腾讯云

综上所述TSF平台，主要是三部分能力：

1.服务治理： 服务治理这部分吸收了Spring Cloud开源框架的服务治理思路。当企业内部微服务化，SOA化后，业务之间的调用，服务注册，服务发现等能有效管理

2.服务生命周期管理：TSF提供虚拟机、物理机、Docker级别的，镜像管理，代码打包上传，应用发布，回滚的能力。与常见的跟虚拟机Autoscaling 的弹性伸缩、容器的CICD有相辅相成的作用。

3.服务监控体系： 微服务间有复杂的rpc、http调用。 腾讯云TSF提供清晰的调用链跟踪分析，业务方可清晰的看到rpc、restful调用的每一个环节的时延，QPS，是否拥塞等

腾讯分布式服务框架（Tencent Service Framework）

常见的自行部署的Dubbo、Spring Cloud 微服务框架，都可以快速切换到**腾讯云TSF平台**，更高效，更稳定，功能更强大！

对比项目	Dubbo	腾讯云TSF
服务注册中心	Zookeeper	Spring Cloud consul 、Zookeeper
服务调用方式	RPC	REST API 、RPC
负载均衡	支持	支持
断路保护	支持	支持
调用链跟踪	无	支持
社区兼容	阿里团队自研	兼容spring cloud 社区
Java应用容器	无	Tomcat
对接消息队列	无	Spring Cloud Stream
代码包管理	无	WAR/JAR/fatJAR
服务扩容	无	自动、手动
代码升级、回滚	无	支持
可用性	可用性一般	99.995%
集群扩容	扩展性一般	理论上无限扩容，支持上万节点同时调用

二、TSF产品使用说明

TSF产品使用说明

产品使用文档主要有以下部分：

1、TSF使用概述

2、JAVA、Spring Boot 容器部署，环境依赖说明

3、应用发布、服务生命周期说明

4、TSF服务注册、发现，调用的使用说明

5、服务调用负载均衡说明

6、熔断器说明

7、TSF全局配置管理介绍

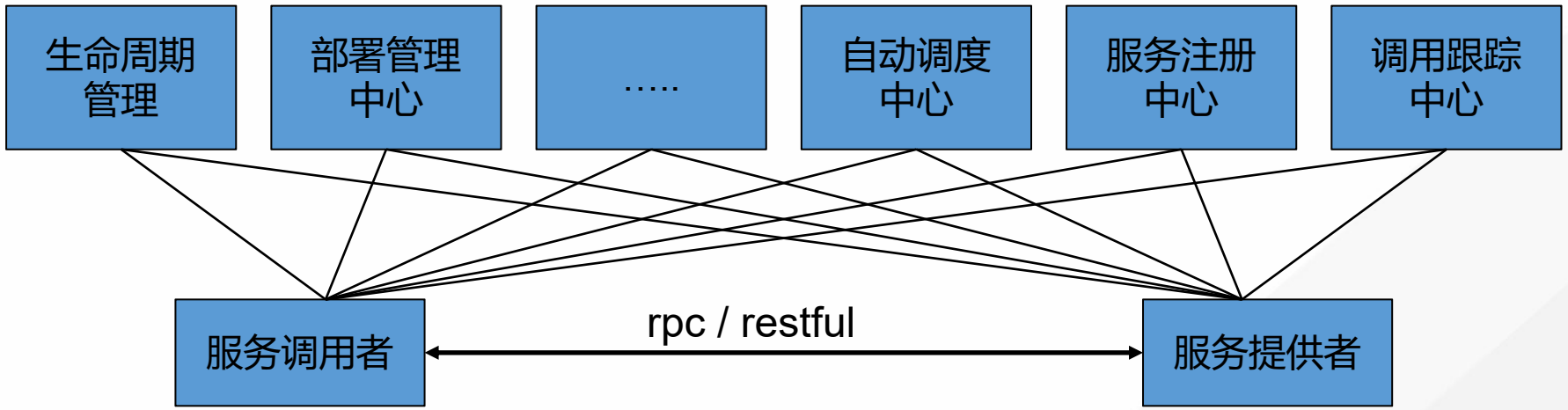
8、监控能力说明

9、产品计费，定价说明

10、存量系统迁移到TSF说明

11、分布式事务能力说明

TSF产品使用概述



步骤1：创建TSF实例：购买具体的TSF实例，分为基础版、高级版，支持扩容

步骤2：常见的1个TSF实例下，会有多个业务模块，具体的模块下有多个服务，每个服务对应多个服务器。层级关系是：TSF实例–》模块—》服务—》具体某台服务器/docker

步骤3：应用服务器锁定：服务部署的第一步是先将云服务器，或IDC的物理服务器，Docker资源，加入某个服务集群里，作为待服务部署的服务器

TSF产品使用概述

步骤4：OSS在具体的应用服务器上安装TSF-Agent,则该服务器后续可使用生命周期管理能力。如上传的代码JAR/WAR包自动的部署安装（按照agent指定的默认路径），扩容、代码回滚、版本变更等能力

步骤5：应用服务器不部署agent也是允许的。但需要开发者自行部署业务，以及会缺少调用链的能力

步骤6：服务部署完毕，发布上线。若作为服务的提供者，则通过spring boot将业务连上spring cloud consul的服务注册中心，供其他业务调用

步骤7：若应用作为服务消费者，则从服务注册中心获取服务提供者的地址。客户端调用时数据流不会经过某中心节点，去中心化的服务治理能力

步骤8：客户端调用跟踪模块向调用跟踪中心上报服务调用记录。TSF-Agent向监控中心上报机器负载情况，自动调度中心分析监控信息并发起扩缩容

TSF产品使用概述

TSF分布式服务框架基于Spring Cloud实现，客户端兼容开源Spring Cloud组件，包含服务注册和发现、服务调用和客户端负载均衡等功能

开发环境：

（1）JDK 1.8、1.7、1.6

（2）Maven，开放指南见https://maven.apache.org/

（3）Spring Boot，开发指南见http://projects.spring.io/spring-boot/

JAVA、Spring Boot 容器部署，环境依赖说明

Spring Boot的目标是让用户更加快速地编写一个用于生产环境的应用:

1、对Spring Framework等其他项目的进行封装

2、新增自动配置机制，让用户可以使用注解，不需要编写大量配置信息

3、嵌入Tomcat、Jetty和Undertow服务器，不需要再部署到其他服务器

4、Spring Boot关注单机的功能实现，没有关注分布式服务

5、腾讯云的Spring Cloud基于Spring Boot提供分布式服务的解决方案

JAVA、Spring Boot 容器部署，环境依赖说明

Spring Boot部署demo：

（1）在用户工程中引入Spring Boot的package

```
<?xml version="1.0"?>
- <project>
  - <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.6.RELEASE</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.tencent.rancho</groupId>
    <artifactId>spring-boot</artifactId>
    <version>0.1.0</version>
  - <dependencies>
    - <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
    </dependencies>
  - <build>
    - <plugins>
      - <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>
</project>
```

JAVA、Spring Boot 容器部署，环境依赖说明

Spring Boot部署demo：

（2）编写应用的代码

```java
package com.tencent.rancho;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class SampleController {

    @RequestMapping("/")
    String home() {
        return "Hello World\n";
    }

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

JAVA、Spring Boot 容器部署，环境依赖说明

Spring Boot部署demo

实现功能：

启动Tomcat服务器，监听localhost:8080，对于访问路径为"/"的请求，响应函数为home，即返回"Hello World”

@RestController表示该类可以处理Web请求的响应

@RequestMapping表示将访问路径映射到类的方法

@EnableAutoConfiguration表示根据引入的依赖包自动配置，引入spring-boot-starter-web表示使用Tomcat和SpringMVC

@RestController和@RequestMapping是Framework已有的注解

@EnableAutoConfiguration是Boot引入的注解

服务生命周期管理

持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。

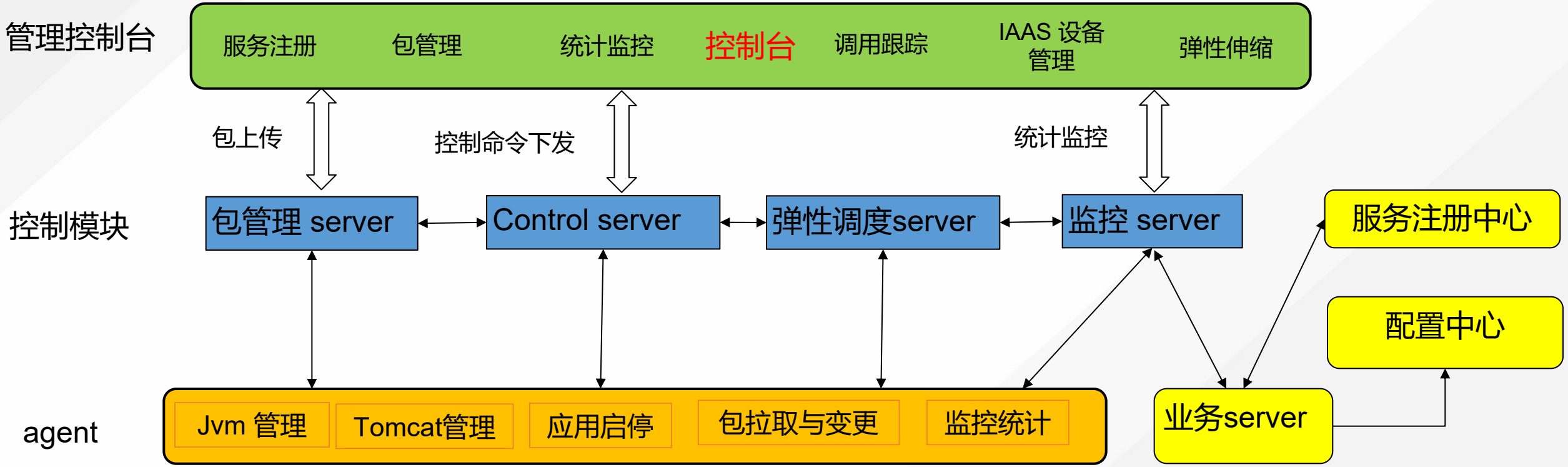
TSF服务发布优势如下：

1、可快速发现软件开发、测试、发布过程中的问题。提供完整的代码发布、回滚，扩容，自动化测试等能力

2、TSF提供一套自动化流程来规范软件的整个生命周期，保证质量可控。

3、降低开发、测试、运维人员的心智负担。

服务生命周期管理



通过控制台在应用机器上安装Agent，将应用部署包上传到包管理中心

通过控制台向控制中心发起扩容操作，控制中心发送扩容指令给Agent

Agent从包管理中心下载应用部署包，将部署包安装到默认路径并启动

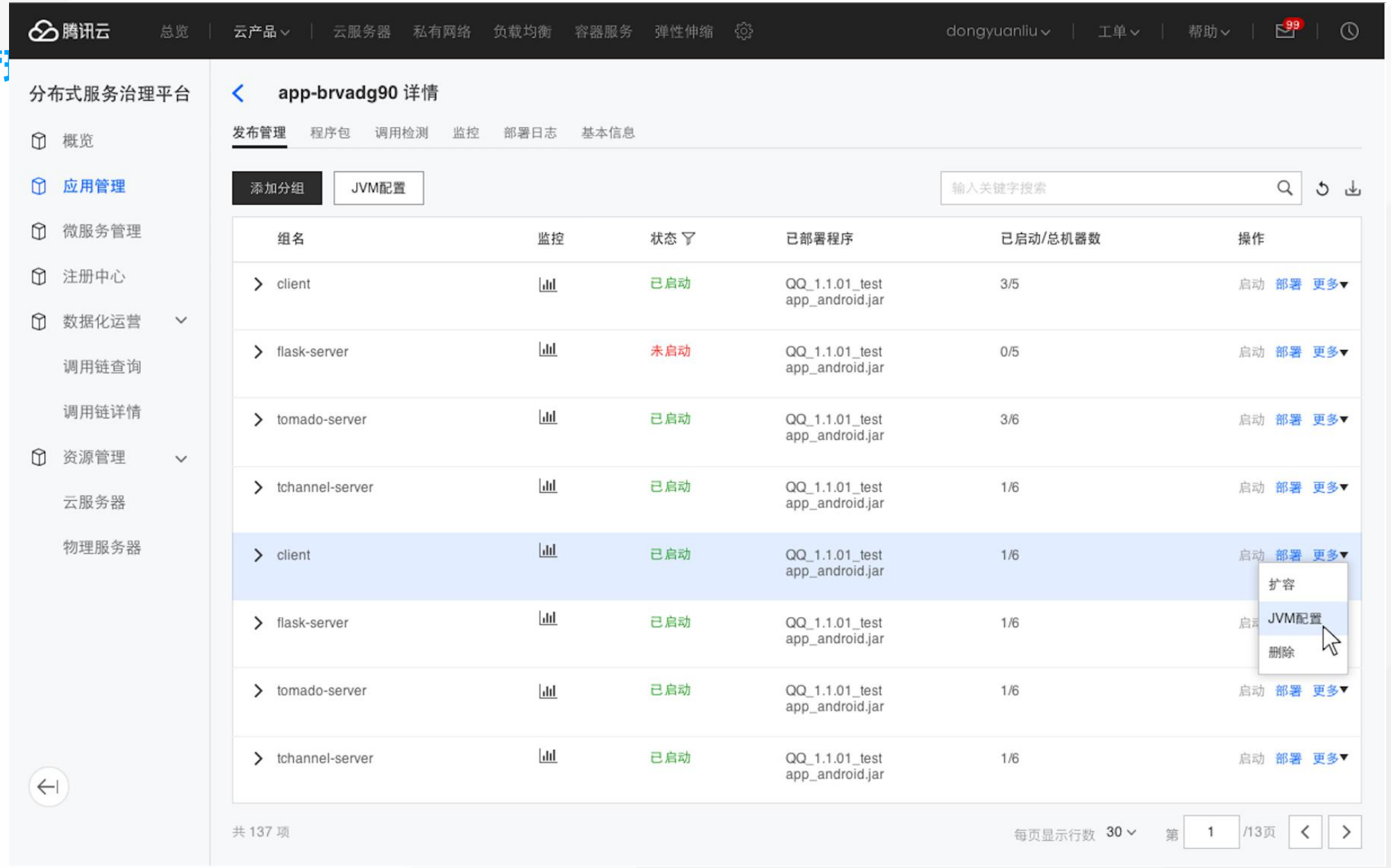
若应用作为服务提供者，则向服务注册中心发起注册

若应用作为服务消费者，则从服务注册中心获取服务提供者的地址

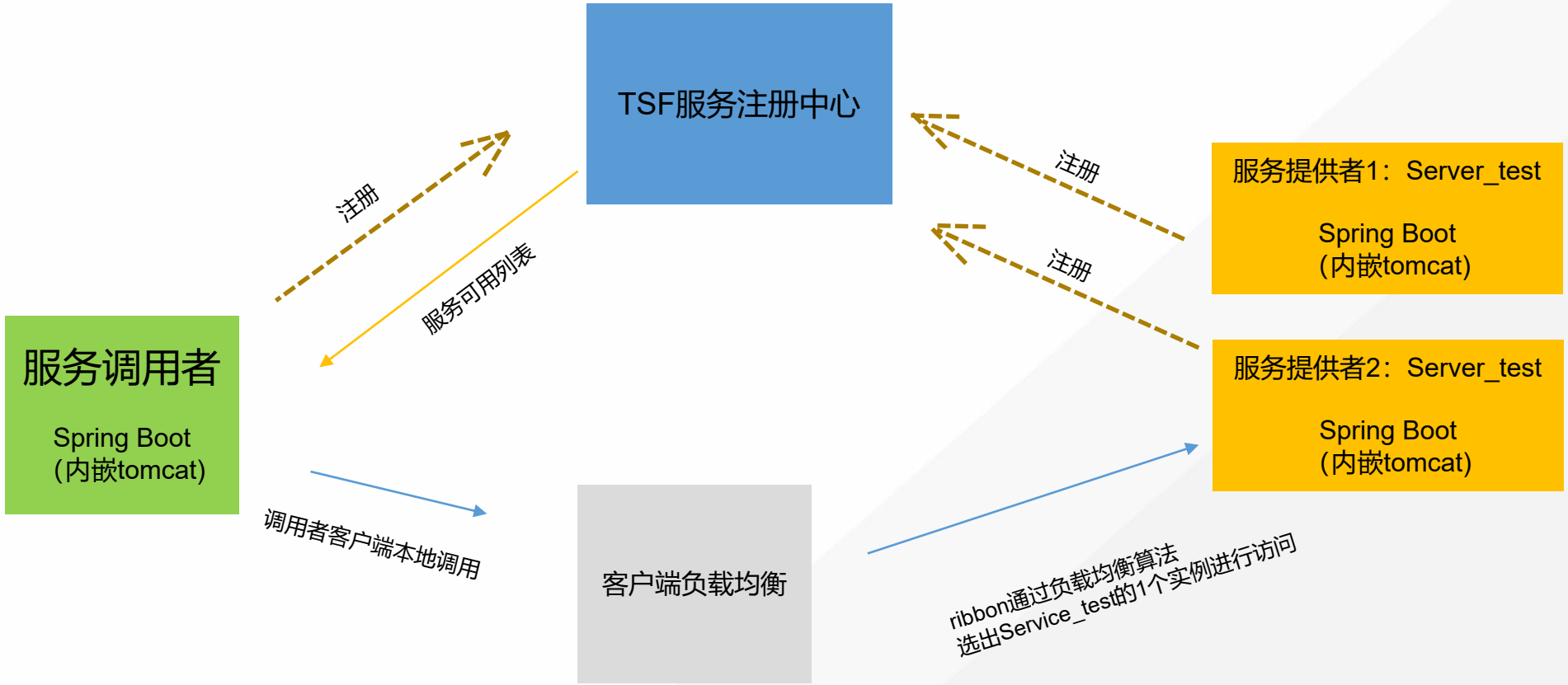
Agent向监控中心上报机器负载情况，自动调度中心分析监控信息并发起扩缩容

服务生命周期管

TSF平台有力的支持服务的自动化的构建（包括代码发布、JVM容器配置、自动化测试）。将服务发布过程中，支持Java的JAR打包上传到TSF平台，提供自动的代码部署服务（注意放通访问策略）



TSF服务注册、发现，调用的客户使用说明



TSF服务注册、发现，调用的客户使用说明

腾讯云TSF服务，基于Spring Cloud Consul提供PaaS化的服务注册中心能力：

注册中心，用于服务端注册远程服务以及客户端发现服务

服务端（服务提供者），对外提供后台服务，将自己的服务信息注册到注册中心

客户端（服务调用者），从注册中心获取远程服务的注册信息，然后进行远程过程调用

TSF服务注册中心 使用说明

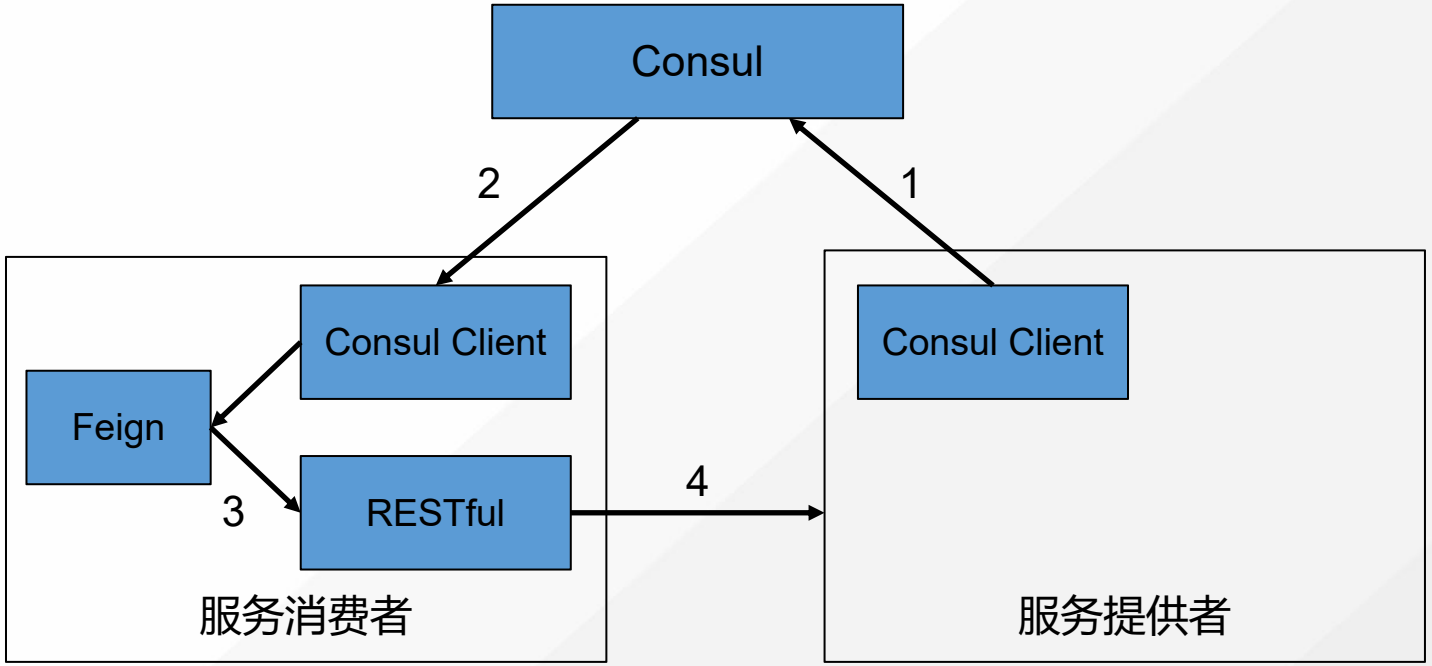
服务注册发现 + RPC调用

1. 服务提供者通过Consul Client向Consul发起注册

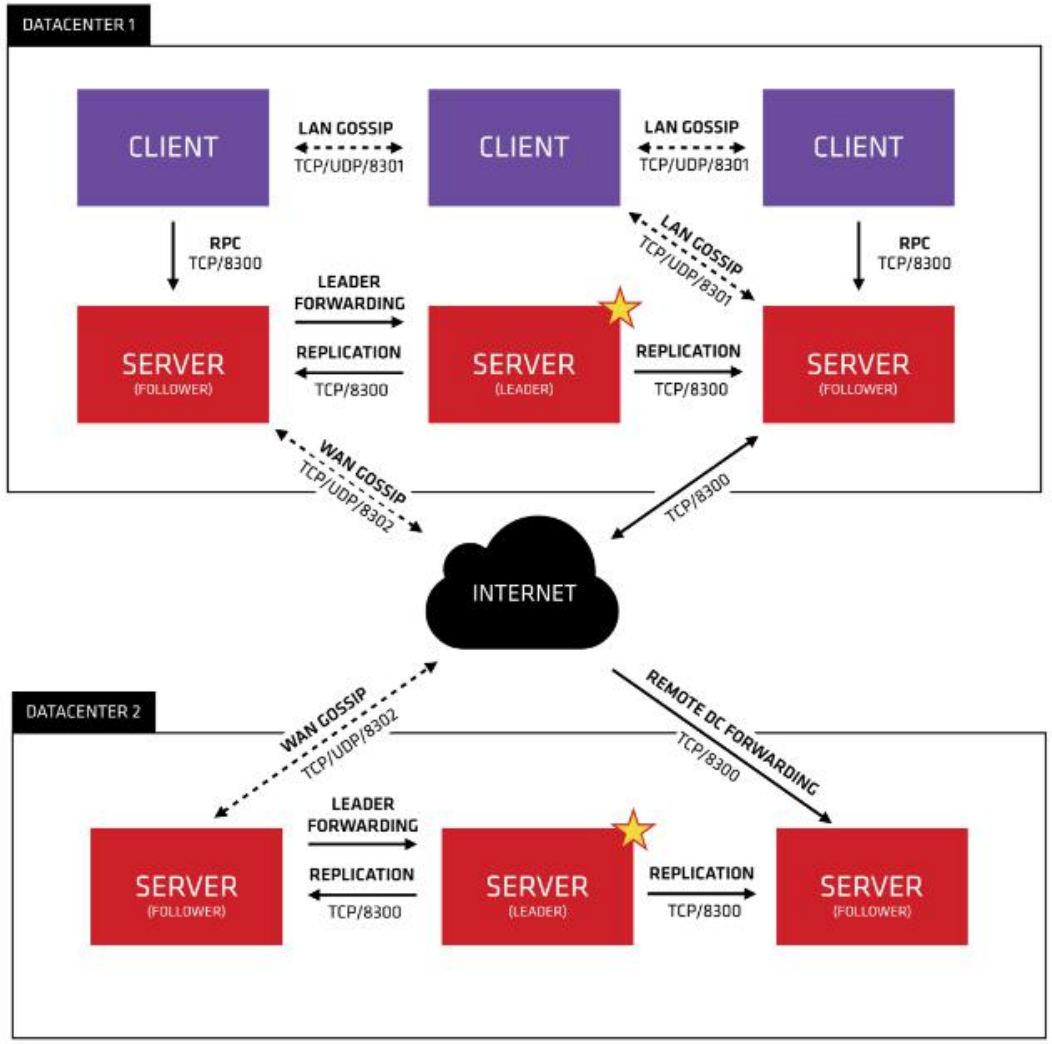
   Consul和Consul Client保持心跳，若服务提供者故障，则将其从服务的地址列表中剔除

2. 服务消费者通过Consul Client从Consul获取服务的地址列表

3. 服务消费者向服务提供者发送RESTful请求



TSF服务注册中心 使用说明

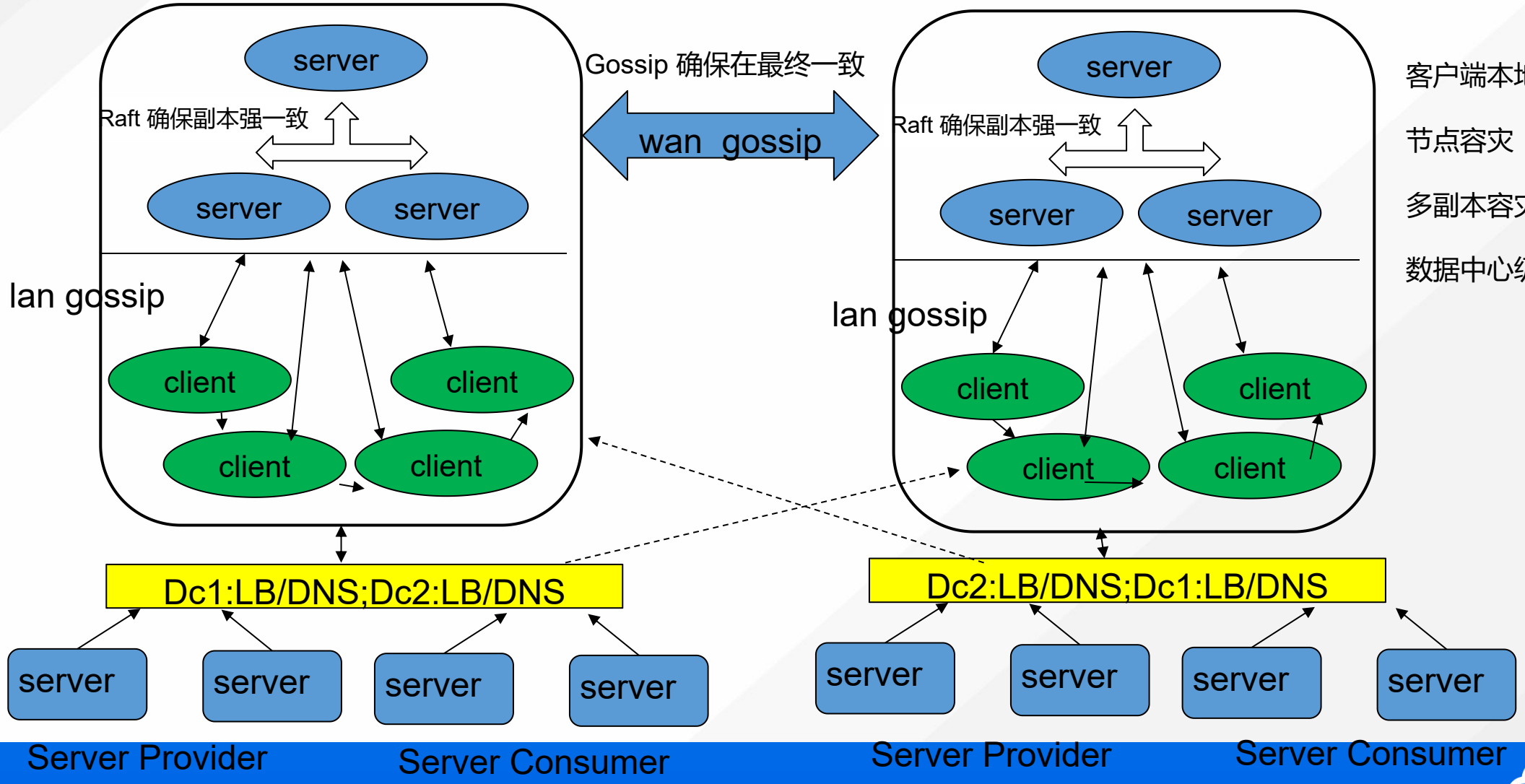


您在本地自行部署Consul服务时，一般分为Client和Server。Client接收应用的请求，将请求转发给Server，可以根据性能要求进行扩展。Server一般3或5台构成集群，负责保存服务注册信息，也可以直接接收应用的请求

腾讯云的TSF平台，通过raft算法，来实现集群内的高可用及数据同步，且可以提供金融级，同城跨数据中心的容灾。每个数据中心的Consul Cluster都会在运行于server模式下的agent节点中选出一个Leader节点，这个选举过程通过Consul实现的raft协议保证，多个 server节点上的Consul数据信息是强一致的

开发者无需关注Consul集群的设备运维，弹性扩缩容，集群一致性同步等。

TSF服务注册中心 使用说明



客户端本地缓存容灾

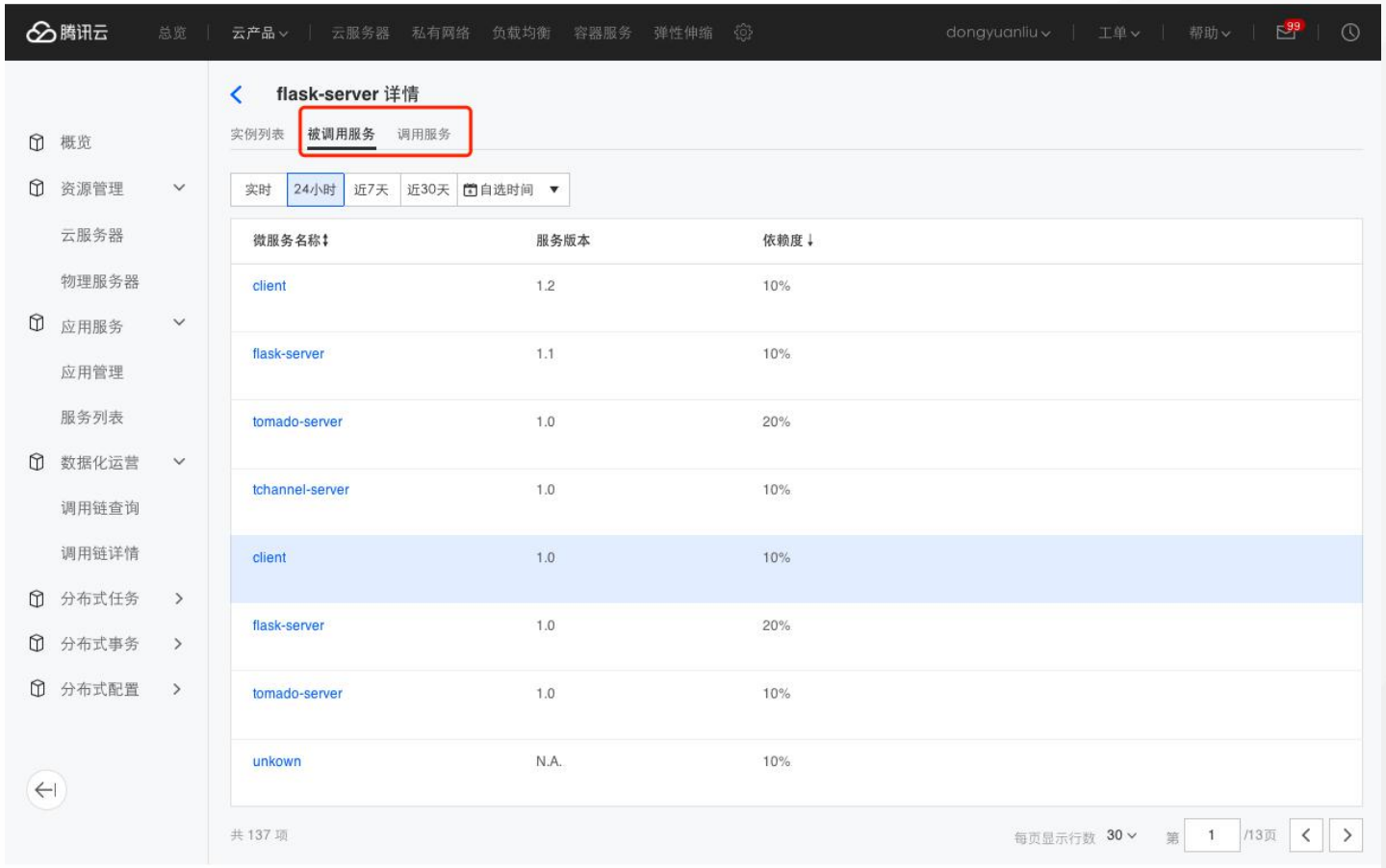
节点容灾

多副本容灾

数据中心级别容灾

TSF服务注册中心 使用说明

如下图所示，服务注册中心的中控平台，可查看到服务调用者，服务提供者的相关信息。（提供spring cloud consul原生版本及腾讯云集成板的web 中控台）



TSF服务提供者 demo演示

服务提供者将本机地址注册到服务注册中心。接收和处理请求，再返回处理结果

在下面示例程序中，服务提供者接收HTTP请求，返回Hello World

开发步骤：

（1）在pom.xml中引入spring-boot和spring-cloud-consul

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-consul-discovery</artifactId>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Dalston.SR2</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>
```

TSF服务提供者 demo演示

（2）编写代码

@EnableDiscoveryClient表示开启服务注册的功能

```
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.boot.autoconfigure.SpringBootApplication;
6  import org.springframework.cloud.client.ServiceInstance;
7  import org.springframework.cloud.client.discovery.DiscoveryClient;
8  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
9  import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.RequestMapping;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import java.util.List;
14
15 @SpringBootApplication
16 @EnableDiscoveryClient
17 @RestController
18 public class ExampleServiceOne {
19
20     @RequestMapping("/")
21     public String home() {
22         return "[Service1-Instance1]Hello World";
23     }
24
25     @RequestMapping("/health/")
26     public String health() {
27         return "[Service1-Instance1]I'm OK";
28     }
29
30     public static void main(String[] args) {
31         SpringApplication.run(ExampleServiceOne.class, args);
32     }
33
34 }
```

TSF服务提供者 demo演示

（3）在application.properties中添加配置

spring.application.name表示服务名称

server.port=8081表示提供服务的端口

spring.cloud.consul.host=10.235.21.35表示服务注册中心的IP

spring.cloud.consul.port=80表示服务注册中心的端口

此时该服务提供者已注册到服务注册中心了

```
1 spring.application.name=Service1
2
3 server.port=8081
4
5 spring.cloud.consul.host=10.235.21.35
6 spring.cloud.consul.port=80
7
8 spring.cloud.consul.discovery.healthCheckUrl=http://10.235.20.24:8081/health/
9 spring.cloud.consul.discovery.healthCheckInterval=10s
```

TSF服务调用者 demo演示

服务调用者从TSF服务注册中心查询某个服务的访问地址，发送请求，获取响应

在大多数情况下，某个服务的请求者同时也是服务提供者

下面示例程序既是服务提供者也是服务请求者。作为服务提供者，接收HTTP请求，进行处理并返回结果。在处理请求的过程中，需要调用服务Service1，因此也是服务请求者

开发步骤：
（1）在pom.xml中引入spring-boot和spring-cloud-consul

```
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.2.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-consul-discovery</artifactId>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Dalston.SR2</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>
```

TSF服务调用者 demo演示

（2）编写代码

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@SpringBootApplication
@EnableDiscoveryClient
@RestController
public class ExampleServiceTwo {

    @Autowired
    private DiscoveryClient discoveryClient;

    @RequestMapping("/")
    public String home() {
        // 从服务注册中心获取服务Service1的访问地址列表
        List<ServiceInstance> serviceDestination = discoveryClient.getInstances("Service1")

        // 从serviceDestination中选择1个访问地址
        // 发送请求，获取响应
        // 其他处理
        // return 处理结果
    }

    @RequestMapping("/health/")
    public String health() {
        return "[Service2-Instance1]I'm OK";
    }

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

}
```

TSF服务调用者 demo演示

（3）在application.properties中添加配置

spring.application.name表示服务名称

server.port=8082表示提供服务的端口

spring.cloud.consul.host=10.235.21.35表示服务注册中心的IP

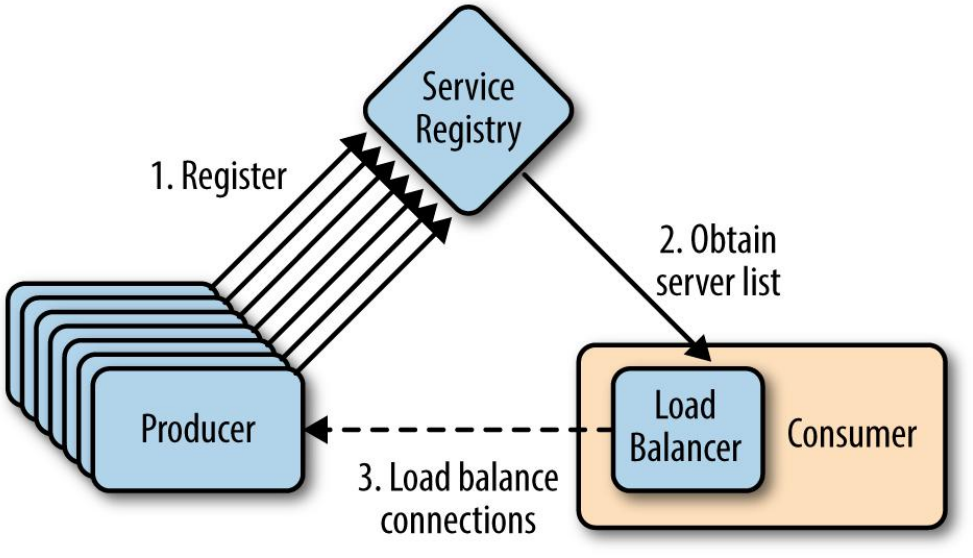
spring.cloud.consul.port=80表示服务注册中心的端口

此时该服务调用者已注册到TSF服务注册中心

```
spring.application.name=Service2

server.port=8082

spring.cloud.consul.host=10.235.21.35
spring.cloud.consul.port=80

spring.cloud.consul.discovery.healthCheckUrl=http://10.235.20.24:8082/health/
spring.cloud.consul.discovery.healthCheckInterval=10s
```

TSF服务调用负载均衡说明

TSF基于Spring Cloud Ribbon/Feign ,提供客户端的软件负载均衡方案，避免服务提供者单点失效，导致整体服务不可用。



TSF服务调用负载均衡说明

负载均衡支持： 轮询 、hash、静态权重、动态权重

负载均衡策略说明	策略描述
BestAvailableRule	选择一个最小的并发请求的server
AvailabilityFilteringRule	过滤掉那些因为一直连接失败的被标记为circuit tripped的后端server，并过滤掉那些高并发的的后端server（active connections 超过配置的阈值）
WeightedResponseTimeRule	根据相应时间分配一个weight，相应时间越长，weight越小，被选中的可能性越低。
RetryRule	对选定的负载均衡策略机上重试机制。
RoundRobinRule	roundRobin方式轮询选择server
RandomRule	随机选择一个server
ZoneAvoidanceRule	复合判断server所在区域的性能和server的可用性选择server

TSF服务调用负载均衡说明

下面示例程序根据第一章节中的服务调用者代码修改而来，在服务Service2调用服务Service1时引入Spring Cloud Feign实现客户端负载均衡

开发步骤：

1. 在pom.xml中引入spring-boot、spring-cloud-consul和spring-cloud-feign

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-consul-discovery</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-feign</artifactId>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Dalston.SR2</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>
```

TSF服务调用负载均衡说明

（2）编写代码

@EnableDiscoveryClient表示开启服务注册的功能
@EnableFeignClients表示开启客户端负载均衡的功能

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.cloud.netflix.feign.FeignClient;

@SpringBootApplication
@RestController
@EnableDiscoveryClient
@EnableFeignClients
public class ExampleServiceTwo {

    @Autowired
    private DiscoveryClient discoveryClient;

    @Autowired
    private RemoteService1 serive1;

    @RequestMapping("/")
    public String home() {

        String info = "[Service2-Instance1]send request to Service1";
        info += "\n[Service2-Instance1]receive response from Service1: ";
        info += serive1.call();

        return info;
    }

    @RequestMapping("/health/")
    public String health() {
        return "[Service2-Instance1]I'm OK";
    }

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

}

// Feign将RemoteService1.call()转换为http://Service1/
// Feign从服务注册中心获取服务Service1的访问地址列表
// Feign实现客户端负载均衡
@FeignClient(value="Service1")
interface RemoteService1 {

    @RequestMapping("/")
    String call();

}
```

TSF服务调用负载均衡说明

（3）在application.properties中添加配置

spring.application.name表示服务名称

server.port=8082表示提供服务的端口

spring.cloud.consul.host=10.235.21.35表示服务注册中心的IP

spring.cloud.consul.port=80表示服务注册中心的端口

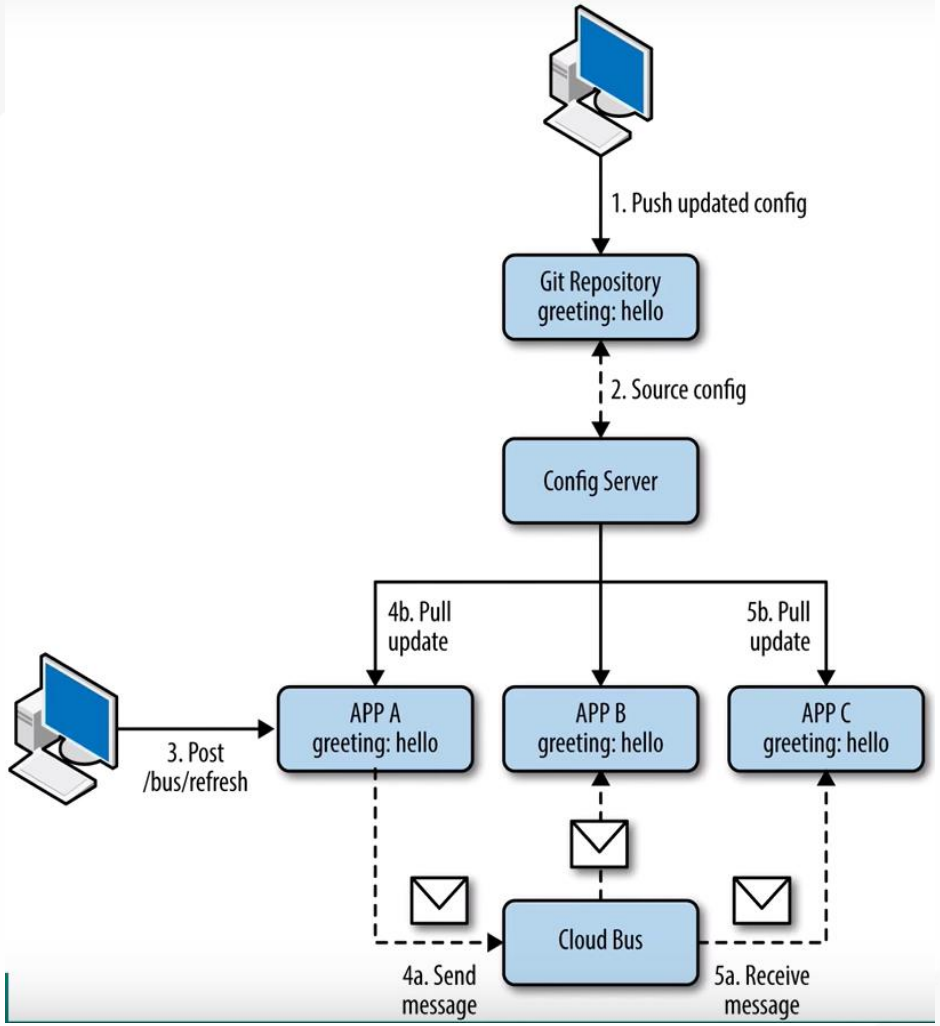
```
spring.application.name=Service2

server.port=8082

spring.cloud.consul.host=10.235.21.35
spring.cloud.consul.port=80

spring.cloud.consul.discovery.healthCheckUrl=http://10.235.20.24:8082/health/
spring.cloud.consul.discovery.healthCheckInterval=10s
```

TSF全局配置推送

在单体式应用中，我们通常的做法是将配置文件和代码放在一起，这没有什么不妥。当你的应用变得越来越大从而不得不进行服务化拆分的时候，会发现各种服务提供者的实例越来越多，修改某一项配置越来越麻烦，你常常不得不为修改某一项配置而重启某个服务所有的服务提供者的实例，甚至为了灰度上线需要更新部分服务提供者的配置。

这个时候有一套配置文件的全局管理方案就变得十分重要，腾讯云的TSF平台结合了SpringCloud Config能力，推出了PaaS化的解决方案

TSF全局配置推送



TSF通过一个轻量级消息代理连接分布式系统的节点。这可以用于广播状态更改（如配置更改）或其他管理指令。

TSF全局配置推送

TSF Config提供基于以下3个维度的配置管理：

应用

这个比较好理解，每个配置都是属于某一个应用的

环境

每个配置都是区分环境的，如dev, test, prod等

版本

这个可能是一般的配置中心所缺乏的，就是对同一份配置的不同版本管理

TSF Config提供版本的支持，也就是说对于一个应用的不同部署实例，可以从服务端获取到不同版本的配置，这对于一些特殊场景如：灰度发布，A/B测试等提供了很好的支持。

TSF 服务容错

分布式系统中经常会出现某个基础服务不可用造成整个系统不可用的情况, 这种现象被称为服务雪崩效应. 为了应对服务雪崩, 一种常见的做法是手动服务降级. 而Hystrix的出现,给我们提供了另一种选择.

雪崩效应带来的影响：

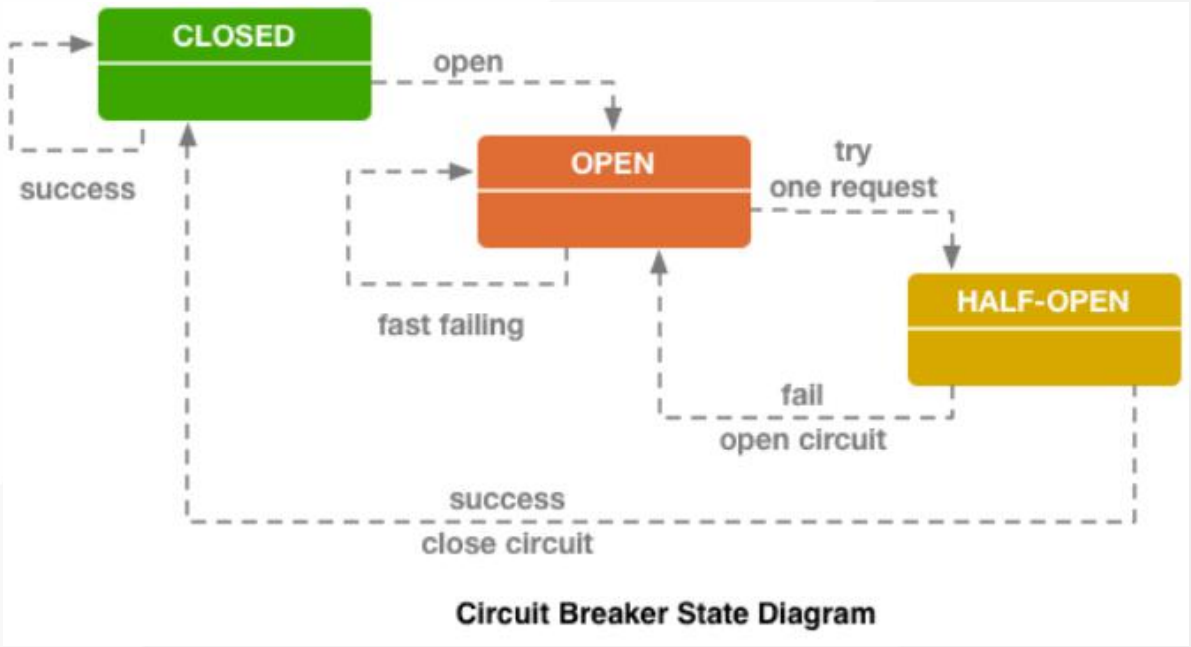
1、单个服务提供者不可用，由于重试的流量加大。导致服务提供者集群不可用

2、服务调用者，由于同步等待造成的资源耗尽，造成服务调用者不可用。

TSF 服务容错

一个应用可能提供多个服务接口，每个服务接口都需要通过RPC调用其他服务若某个被调用服务的实例全部故障，则可能造成大量线程等待，导致整个实例无法正常服务。为每个服务设置Fallback，当某个服务故障时，则直接调用Fallback，防止雪崩：

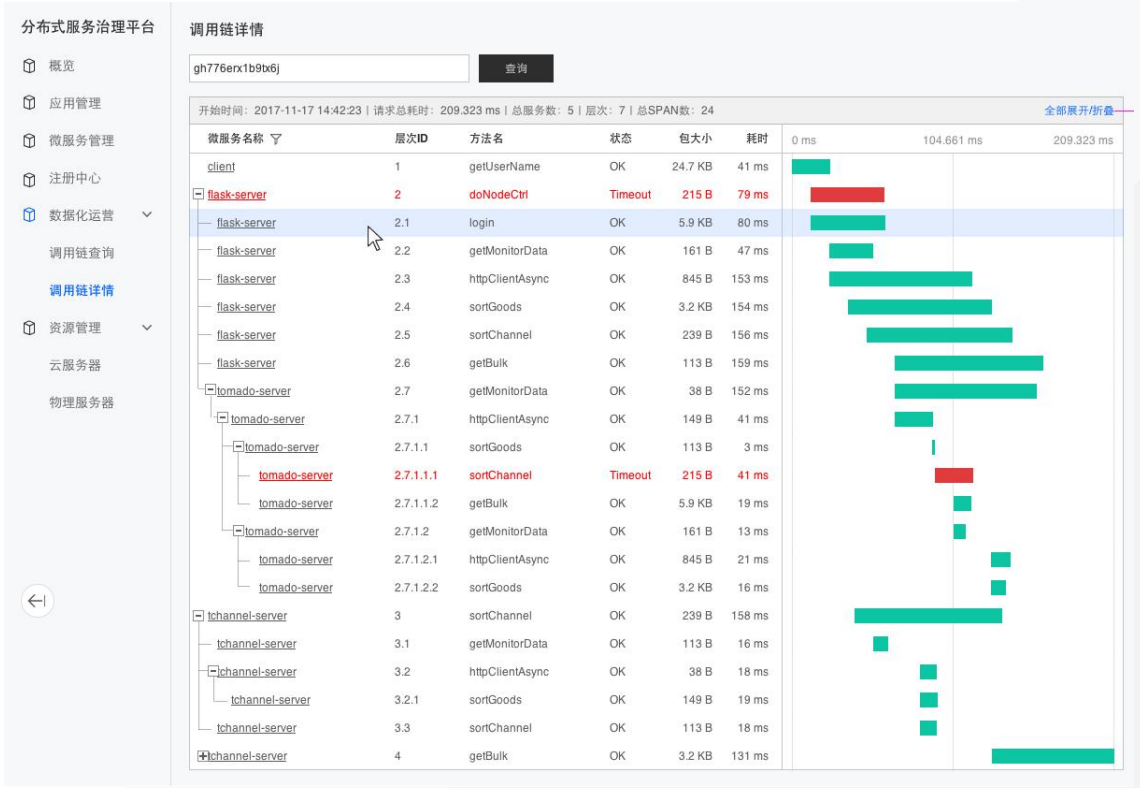
1、断路器统计调用每个服务的失败率
2、若调用某个服务的失败率超过一定阈值，则触发断路。避免有大量的同步调用的线程，阻塞于此
3、对于触发断路的服务请求，绝大部分不再发起远程调用，直接调用Fallback
4、对于触发断路的服务请求，选择少量发起远程调用，测试该服务是否恢复
5、若该服务恢复，则关闭断路



Circuit Breaker State Diagram

TSF监控说明

TSF平台提供分布式调用跟踪能力，可清晰定位到具体的**业务瓶颈**。



TSF监控说明

核心数据结构：

TraceID：用来标识每一条业务请求链的唯一ID，TraceID需要在整个调用链路上传递：机器IP+进程ID+递增序列号 保证唯一且有序

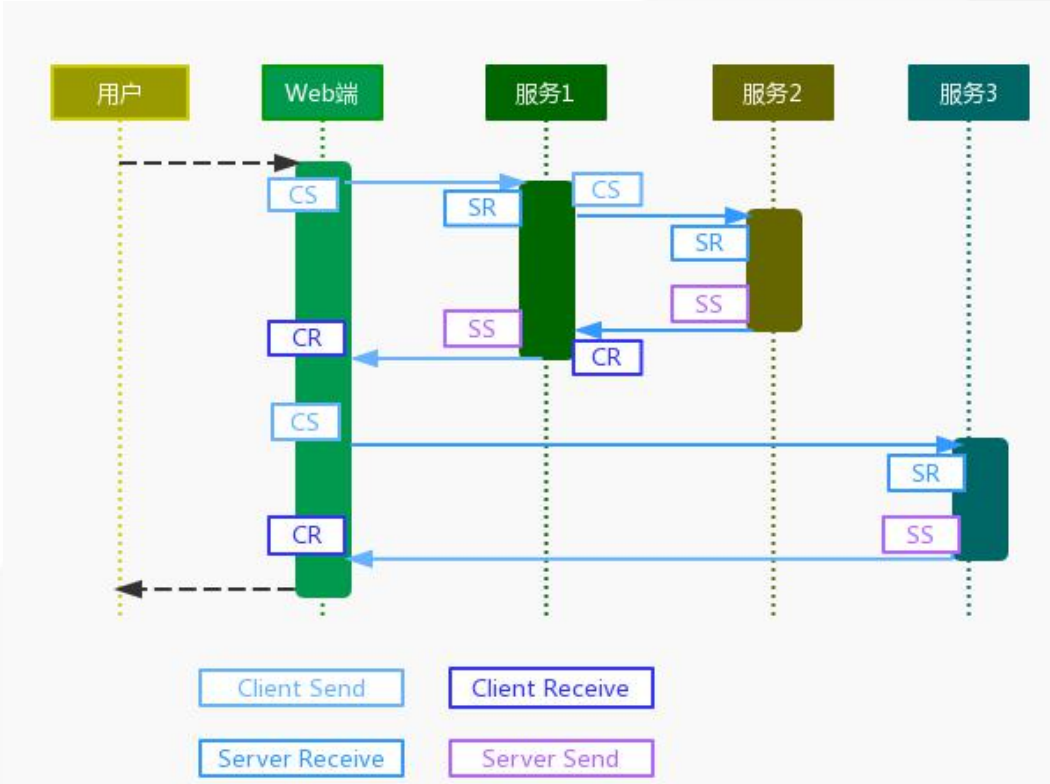
Span：请求链中的每一个环节为一个Span，每一个Span有一个

SpanId来标识，前后Span间形成父子关系

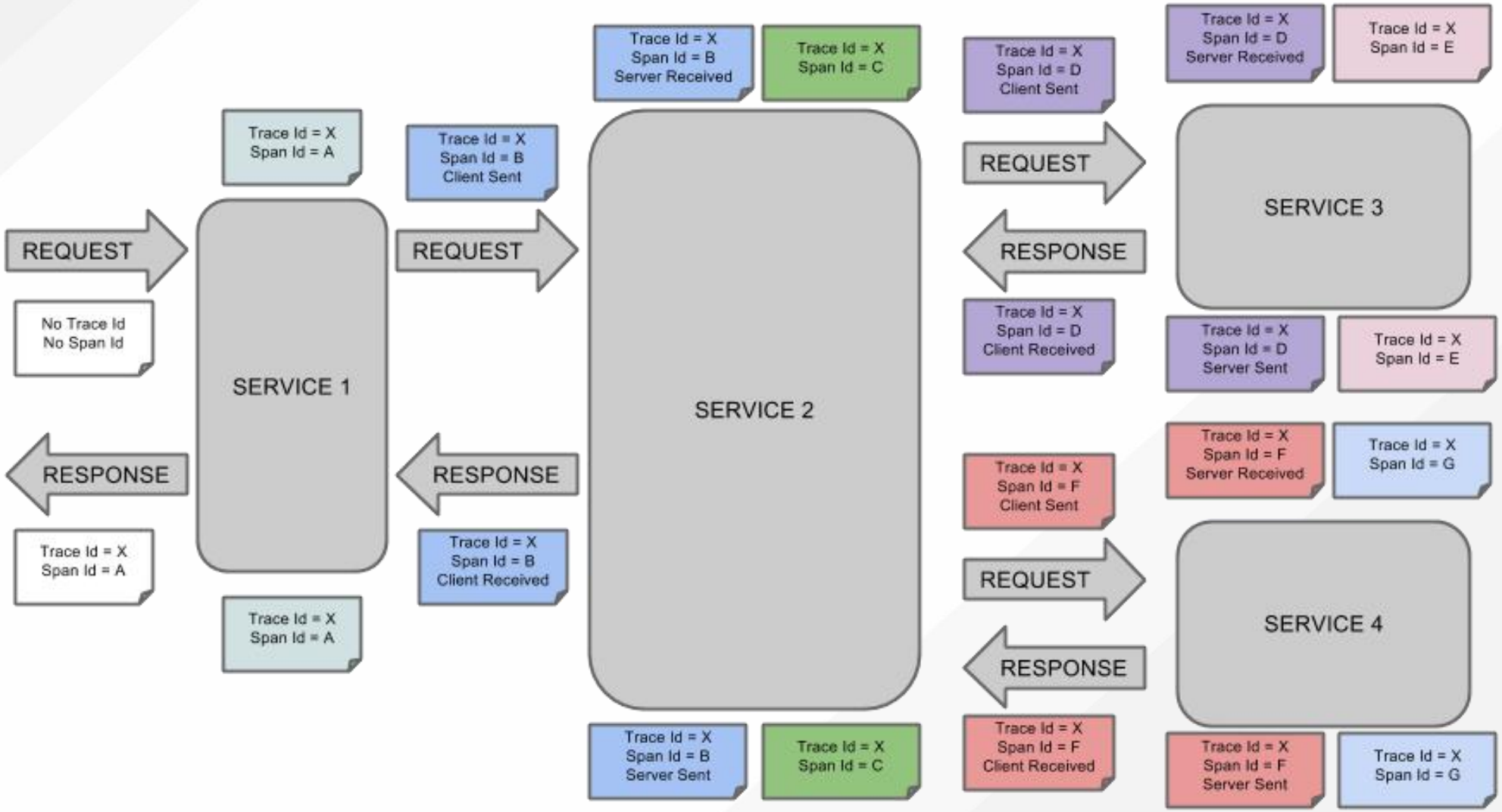
Annotation：业务自定义的埋点信息，可以是sql、用户ID等关键信息

如何对业务透明，如何在上下文中记录：
底层框架支持，无需用户干预
框架埋点四个阶段：

1、Client Send：
2、Server Receive：
3、Server Send：
4、Client Receive：

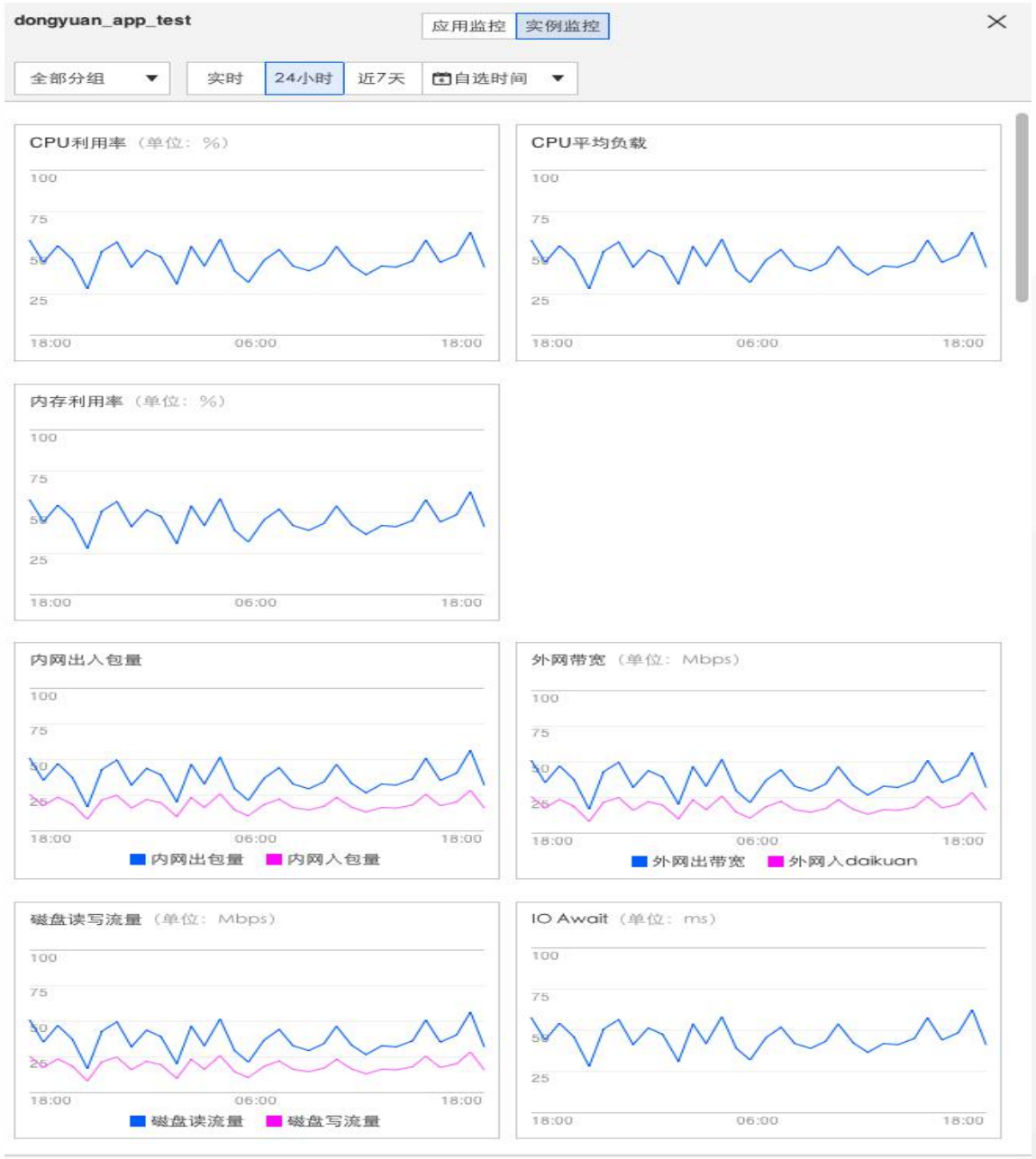


TSF监控说明



TSF监控说明

针对具体服务集群（或具体服务），提供详尽的服务器级别（CPU内存磁盘、进程）等的监控数据统计、告警



Dubbo迁移到Spring Cloud概述

用户原来使用Dubbo，需要使用spring cloud，改造工作量如下

第一步，改造Dubbo，支持Spring Boot的自动配置机制，兼容Spring Cloud

第二步，用户需要修改代码和配置

Dubbo/HSF代码风格： 改造有较大难度

```
import org.springframework.context.ApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

import com.tencent.test.SampleService;

public class HsfServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        ApplicationContext ctx=WebApplicationContextUtils.getWebApplicationContext(req.getServletContext());

        SampleService sampleService = (SampleService) ctx.getBean("sampleService");

        resp.getWriter().println(sampleService.echo(Long.toString(System.currentTimeMillis())));
    }
}
```

Dubbo迁移到Spring Cloud概述

腾讯云

腾讯云的TSF服务，对spring cloud 进行改造后，服务注册发现的接入，仅需简单修改，逻辑代码无任何变动，即可迁移上云。上云“零”成本！

TSF接入兼容dubbo：

1、替换业务发布包中的dubbo-registry-zookeeper.jar包替换为我们的tsf-registry-consul.jar包即可

2、修改dubbo 配置项：将address部分替换成我们提供的url即可使用腾讯TSF服务

```
<dubbo:registry id="xxx1" address="xxx://ip:port" /> <!-- 定义注册中心 -->
```

最佳实践 – 兼容原有应用

在微服务改造和迁移过程，如何接入原有应用，让原有应用和微服务应用并存，是一个重要的问题

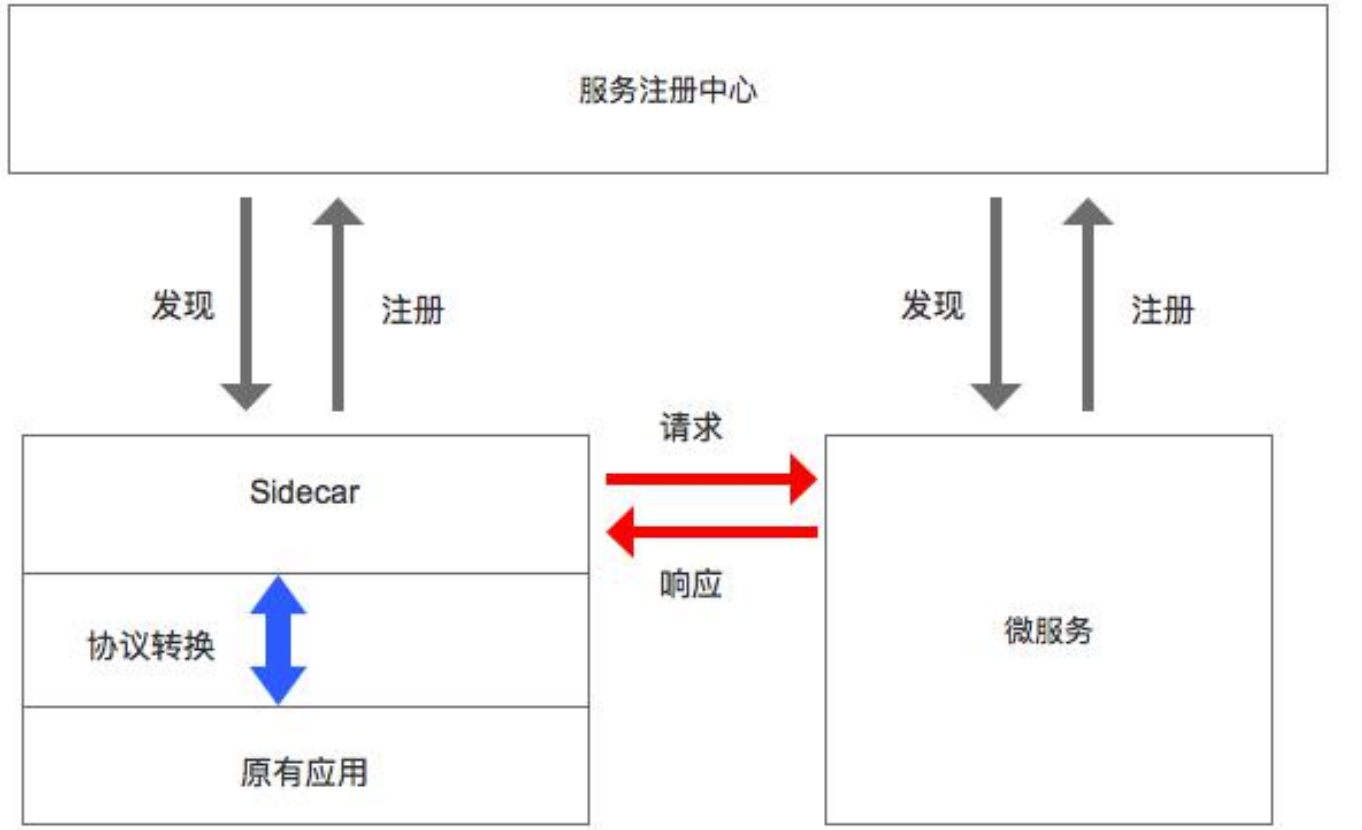
TSF提供基于Sidecar的解决方案，让原来应用无需改动一行代码，即可接入微服务平台

主要优点：

（1）将原有应用接入服务注册中心

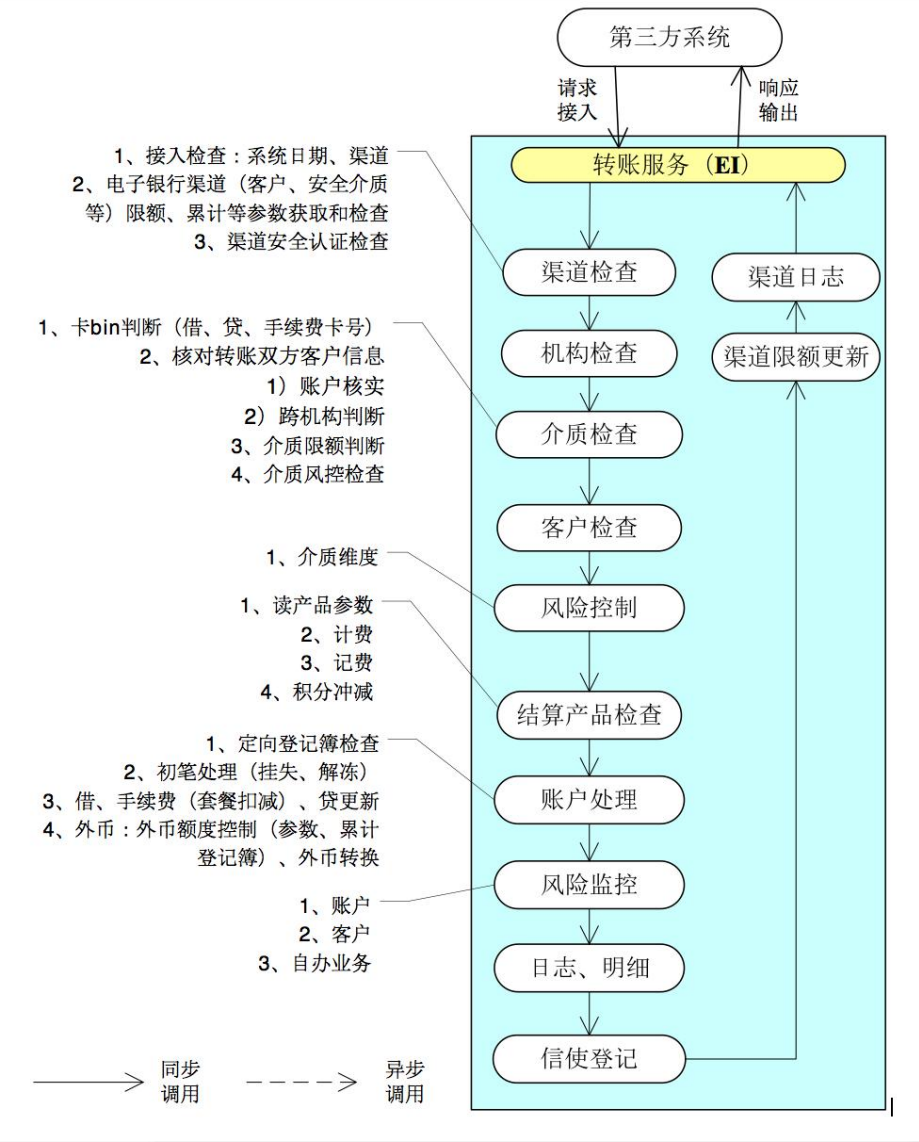
（2）通过微服务框架访问原有应用， 不需要业务代码考虑负载均衡和断路保护等

（3）将原有应用纳入微服务平台的监控系统中



TSF-分布式事务能力介绍

以个人转账交易为例，下图为一个标准的银行业务转账流程，该事务为同步转账事务，过程涉及流程较多。服务框架需支持分布式的事务处理能力：



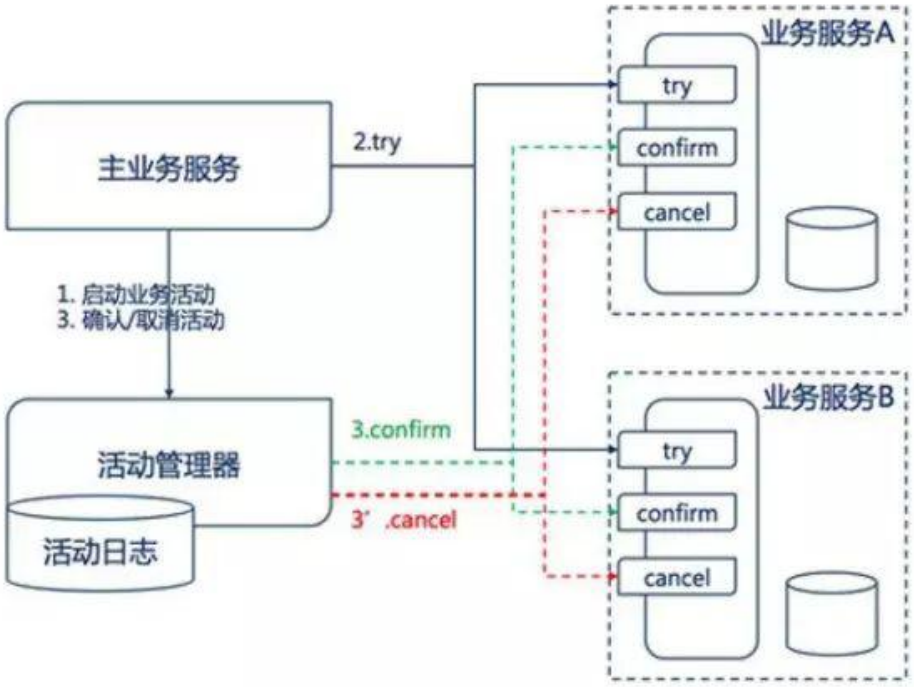
TSF-分布式事务能力介绍

腾讯云TSF框架，提供基于TCC（Try-Confirm-Cancel）的事务方案，来解决跨服务的一致性问题

第一阶段：主业务服务分别调用所有从业务服务的 try 操作，并在活动管理器中记录所有从业务服务。当所有从业务服务 try 成功或者某个从业务服务 try 失败时，进入第二阶段。

第二阶段：活动管理器根据第一阶段从业务服务的 try 结果来执行 confirm 或 cancel 操作。如果第一阶段所有从业务服务都 try 成功，则协作者调用所有从业务服务的 confirm 操作，否则，调用所有从业务服务的 cancel 操作。（confirm操作，需要业务满足幂等）

在第二阶段中，confirm 和 cancel 同样存在失败情况，所以需要对这两种情况做 异常处理以保证数据一致性。
Confirm 失败：则回滚所有 confirm 操作并执行 cancel 操作。
Cancel 失败：从业务服务需要提供自动 cancel 机制，以保证 cancel 成功。（cancel操作，需要业务满足幂等）

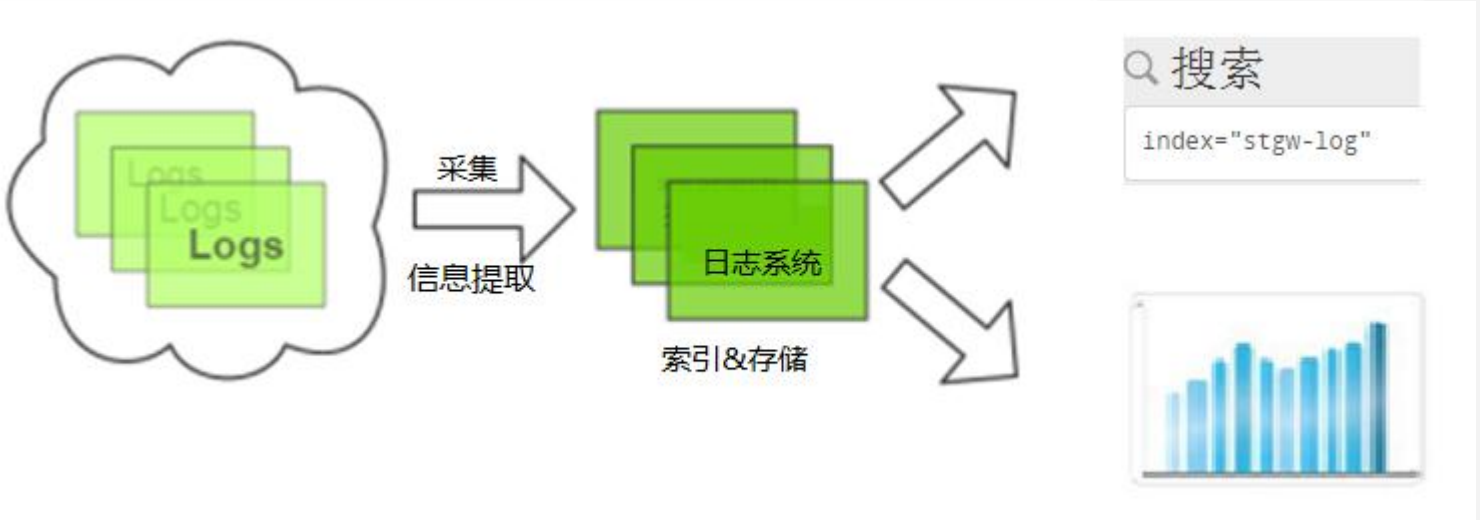


TSF-日志服务介绍

腾讯云基于ElasticSearch自研的日志服务，目前包括腾讯云CLB，云监控等产品均使用该日志服务。

TSF 日志服务具有如下特色功能：

- 查看错误日志的上下文环境
- 使用逻辑表，底层自动管理Index滚动
- 支持SQL的查询接口
- 支持转储，如转储到COS



轻量部署

功能点	数量	形式	配置	是否必备
基础环境	6台	虚拟机/物理机	4核8G radhat 7.2	（必备）
应用部署	3~5台	虚拟机/物理机	核8G radhat 7.2	（业务可选）