



腾讯云

DSGP分布式服务治理平台 产品使用文档

HoraceZhang

2017.08.08

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

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

业界优势

- 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密钥管理、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

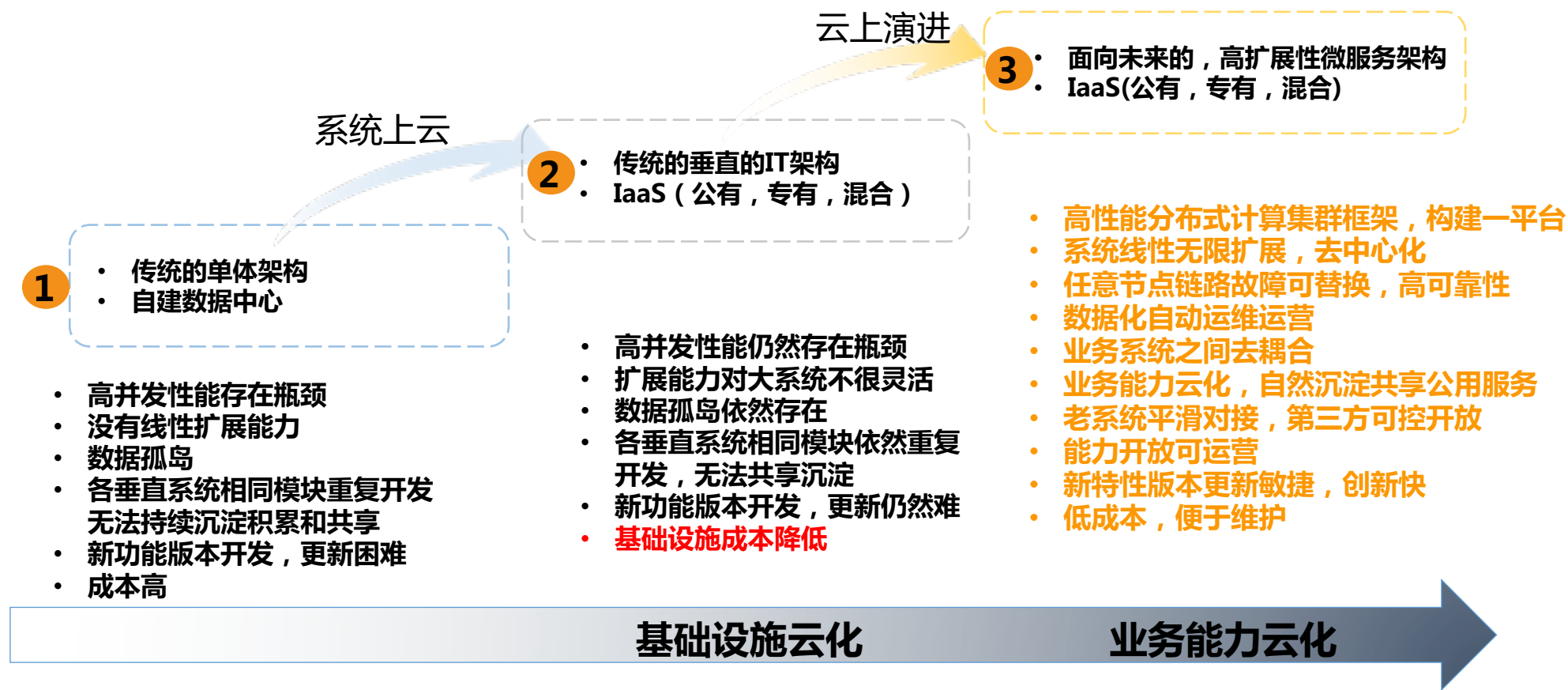
负载均衡、
API网关

加密机、
密钥管理服务

无服务器
函数计算

HOT

一、分布式服务治理平台 概述

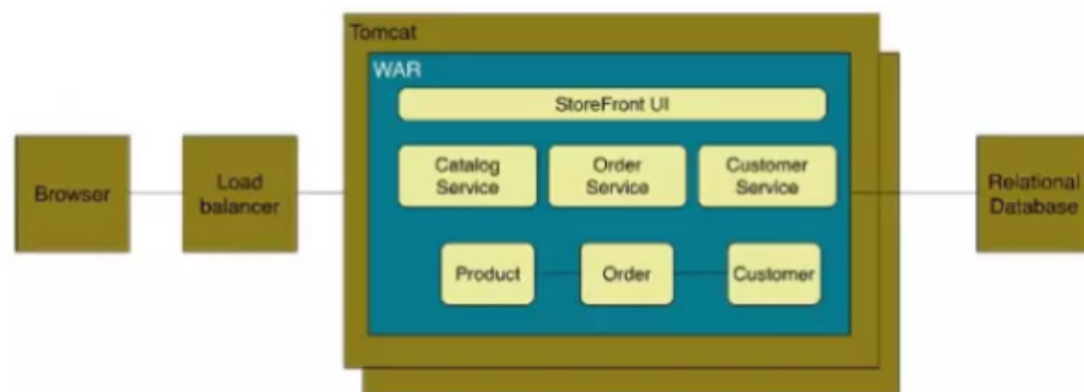


云计算时代，企业信息化演进不仅仅是把IT系统搬到云上，而是让业务与信息系统深度融合，改变业务运营和创新模式。

单体架构:

假设你正在构建一个在线商店系统：客户下订单、核对清单和信用卡额度，并将货物运输给客户。整个应用部署在tomcat容器里。软件设计遵循MVC的软件设计思想。

很快，你们团队一定能构造出如右图所示的系统，这种将所有功能都部署在一个web容器中运行的系统就叫做单体架构



单体架构的特点、好处:

- 单一代码库、IDE友好
- 容易部署，单一WAR/JAR包
- 开发模型简单，一份代码库进行编码、构建和部署
- 简单可伸缩性，在负载均衡器后运行多份应用副本
- 技术栈单一

单体架构的存在问题:

- 庞大的代码库，关系错综复杂
- 译构建时间很长，与持续集成配合麻烦
- 部署臃肿，影响部署速度与频次
- 扩展能力与弹性受限
- 新技术与工具框架使用会受限

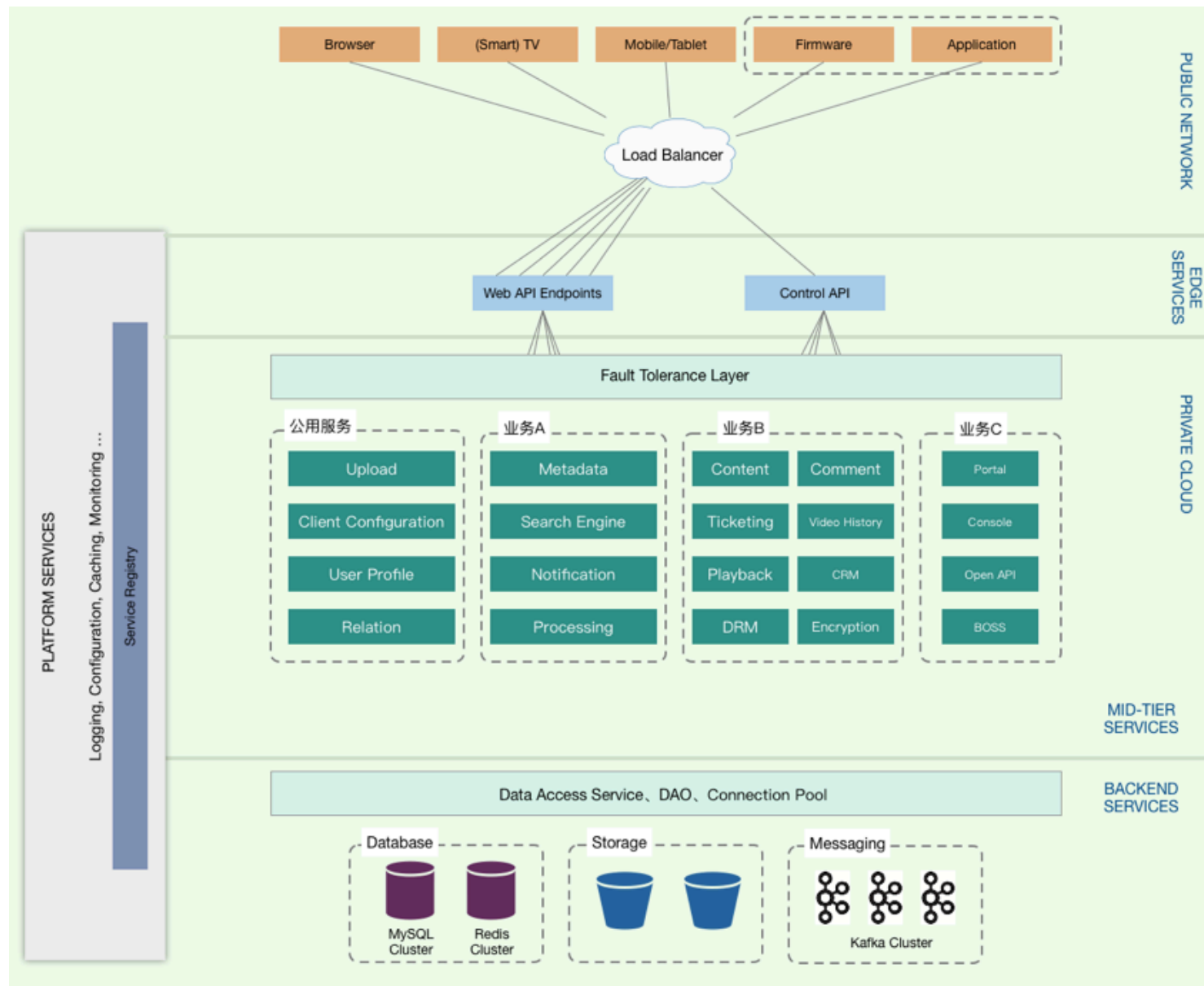
SOA架构:

对业务进行解耦，通过Pub-Sub或RPC进行服务间调用关系解耦

服务独立性，多数服务可以进行独立打包、发布

通常每个服务的技术栈单一

部署简单，具备可伸缩性



SOA架构、存在问题：

对于部分服务而言，代码库依然巨大

打包、发布、部署流程不够好

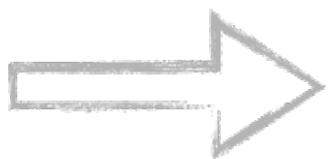
可伸缩性变强，但依然不够好

维护团队间沟通受阻，技术经验有效传递不够

服务增多对开发人员不够友好



单体架构



SOA



Microservices

塔式应用、单体应用，扩展困难，维护困难

大型系统分层、解耦，标准接口调用，分布式系统

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

微服务架构、与SOA架构的对比：

对比项	SOA	微服务
组件大小	大块业务逻辑	单独任务或小块业务逻辑
耦合	通常松耦合	松耦合
使用场景	传统企业上云	云上最新一代架构
管理	中央管理中心	去中心化管理
目标	避免传统单体服务弊端	高扩展，满足快速的业务迭代开发需求。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。可独立自主的进行业务部署、扩展

微服务架构优势 - 业务功能独立、代码库独立：

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

例如，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

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

微服务架构优势 - 技术栈独立:

每个微服务都有自己独立的技术栈实现

根据业务实现需求来选择最合适的技术栈

团队可以尝试新的技术、工具或框架

所选技术栈一般来说都很轻量级

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

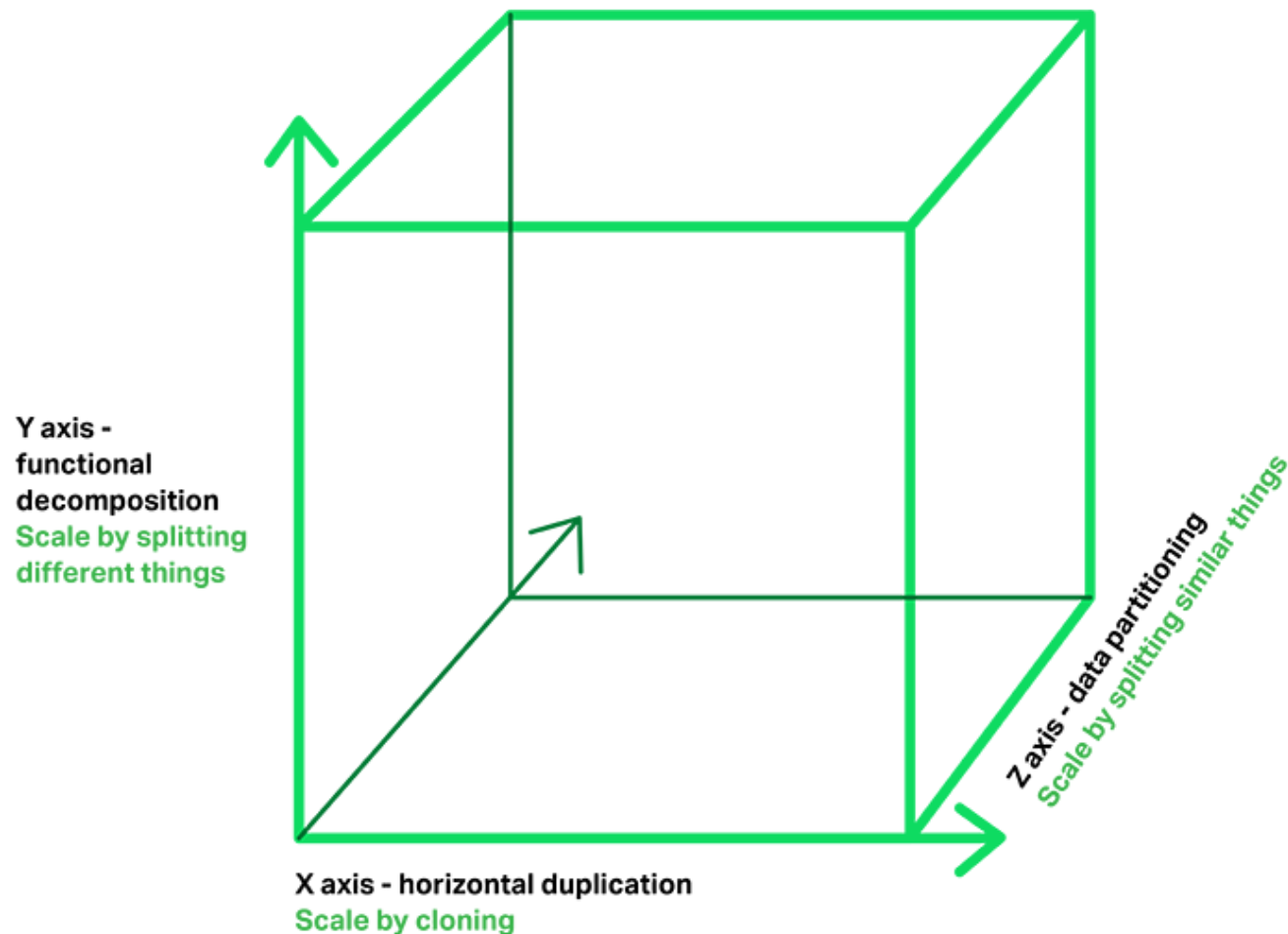
无需引入未被使用的技术或库

微服务架构优势 - 独立的可伸缩性:

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



微服务架构，存在的问题？

一个不足之处在于，微服务应用是分布式系统，由此会带来固有的复杂性。开发者需要在 **RPC** 调用/**restful**调用 或者消息传递之间选择并完成进程间通讯机制。此外，他们必须写代码来处理消息传递中速度过慢或者不可用等局部失效问题。

另外一个挑战在于，微服务架构模式应用的改变将会波及多个服务。部署一个微服务应用也很复杂，一个单体应用只需要在复杂均衡器后面部署各自的服务器就好了。每个应用实例是需要配置诸如数据库和消息中间件等基础服务。每个服务都有多个实例，这就形成大量需要配置、部署、扩展和监控的部分。除此之外，你还需要完成一个服务发现、管理机制，以用来动态发现与它通讯服务的地址。

最后，微服务模式下，客户端的**request**、**respond** 通常会经过数个、甚至数十个模块，调用。一旦出现故障定位问题复杂，日志流水分散在各自的微服务模块下，难以聚合。

怎么办呢？

企业架构的去中心化、微服务化是大势所趋！腾讯云能协助您平稳的演进云端架构。



腾讯成熟中间件

- 腾讯内部多年验证的中间件服务，稳定可靠



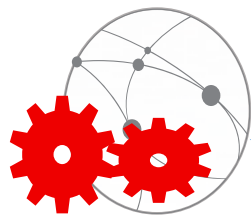
微服务化架构

- 服务原子化解耦，去中心化的服务调用、注册、服务治理能力
- 高性能RPC、RESTful框架



自动化,高可靠

- 任何节点和链路故障情况，能够自动检测，优化确保高可靠性



异步化，最终一致

- 高性能消息服务框架
- 系统应用尽量无状态化
- 确保系统最终一致



数据化运营

- 服务运行实时监控，数据积累可视化
- 数据积累提供系统的优化基础



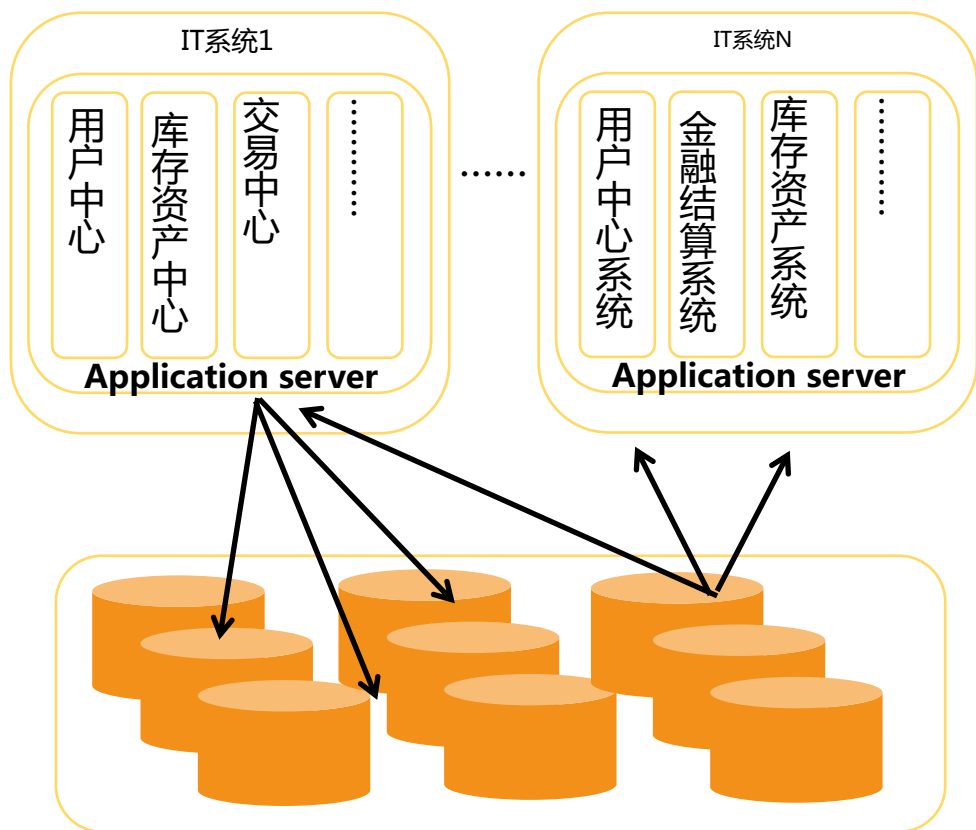
去中心化，水平扩展

- 服务能力，随着资源加入，微服务级线性的性能和容量扩展

腾讯云分布式服务治理平台 (Distributed Service Governance Platform)

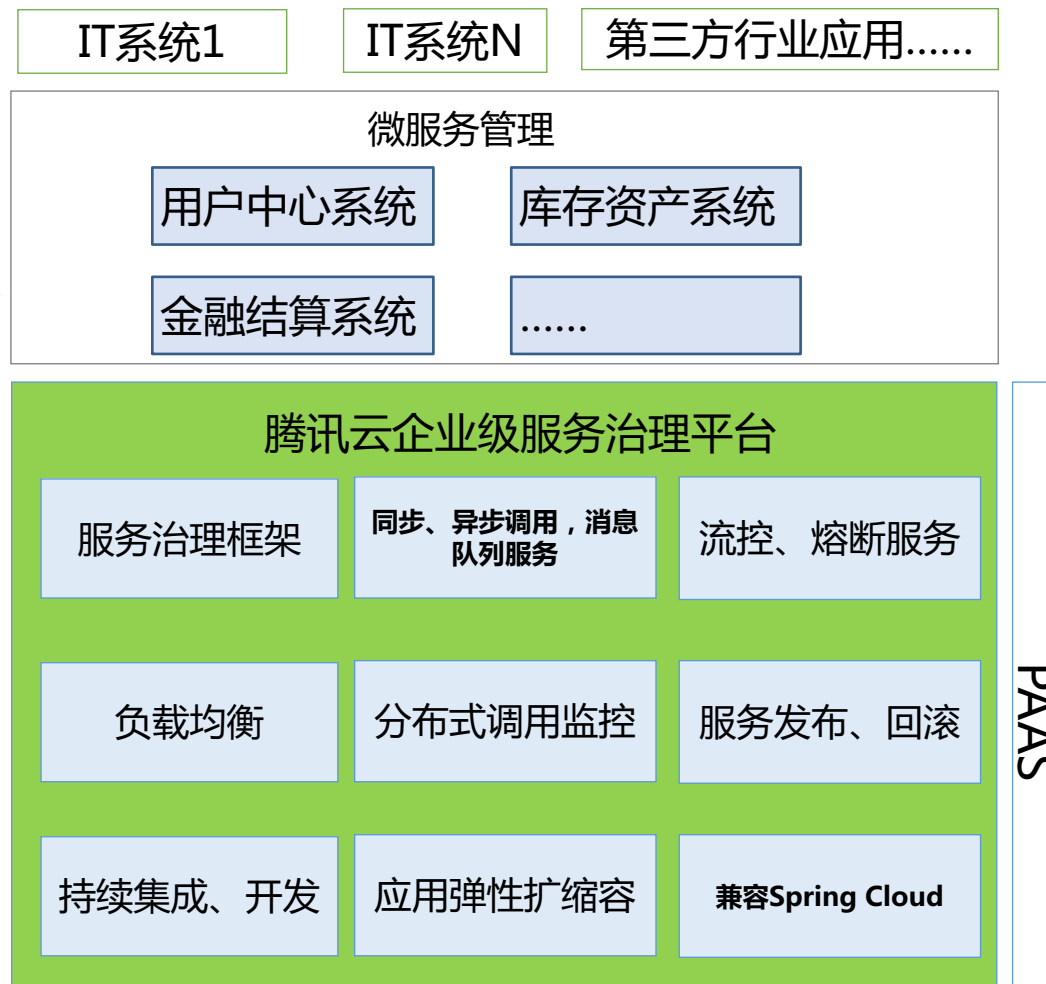


腾讯云分布式服务治理平台，是一个围绕服务注册发现、管理、微服务、服务持续集成的PaaS平台，提供多样的应用发布能力和轻量级微服务解决方案



微服务化的金融云平台

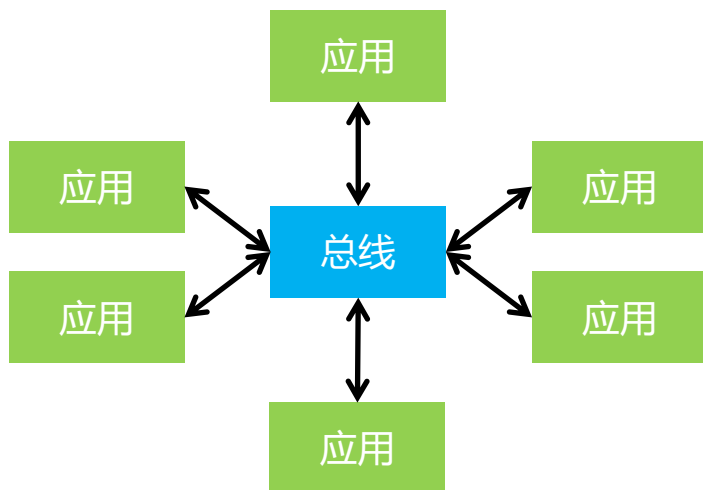
完善的服务发现、注册、调用、代码发布管理



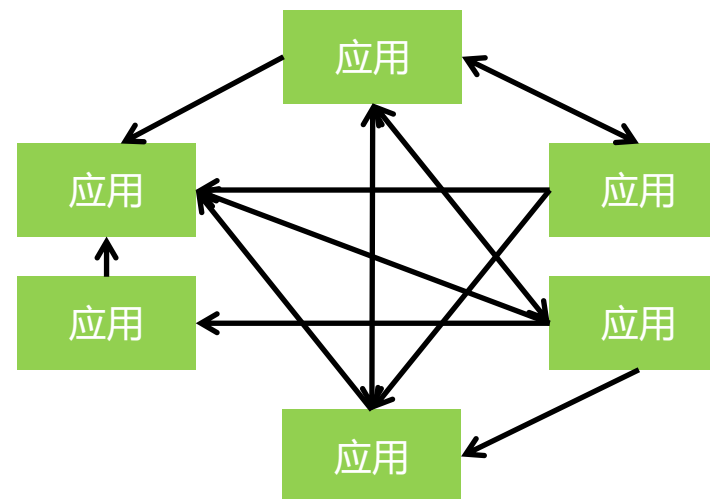
DSGP基于开源的Spring Cloud框架，提供全套的服务治理、服务生命周期管理，调用链跟踪等服务。Java开发者迁移上云“零门槛”！



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



中心化 VS 去中心化

1. 中心化

以IBM ESB (Enterprise Service Bus, 企业服务总线) 或者消息总线作为中心点; 通过消息队列来做集中式网络传输通道, 重架构形式, 方便管理、消息审计。

2. 去中心化

服务划分越来越细, 个数越来越多, 新功能迭代越来越快, 如何对成千上万的服务进行智能管理? 如何快速的变更、上线服务, 国内阿里Dubbo、腾讯Tars, 国外以Spring Cloud为主, 插件形式, 功能齐全, 国内越来越多互联网公司开始使用。

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。

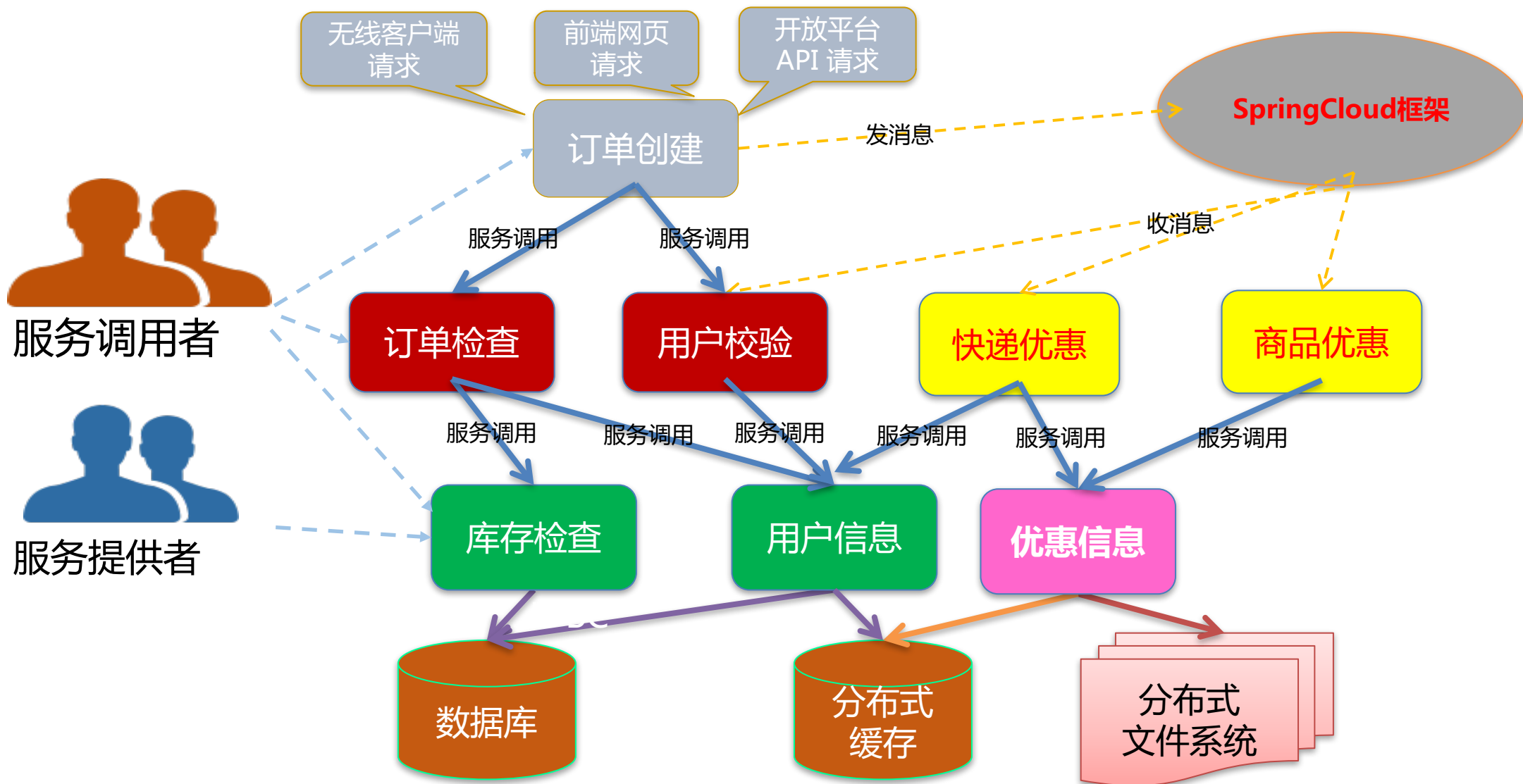
DSGP提供了一个高性能的 restful/rpc 调用 框架，能够构建高可用的分布式系统，系统地考虑各个应用之间的分布式服务发现、服务路由、服务调用以及服务安全等细节



腾讯云分布式服务治理平台 (Distributed Service Governance Platform)



去中心化服务治理，DSGP提供开源Spring Cloud框架的PAAS化服务，上手即用，免除运维烦恼



微服务框架核心其二：服务自动化部署

1、运维自动化，动态故障剔除，负载均衡

2、水平扩展自动化，高效弹性伸缩

3、持续集成：持续集成是一种软件开发实践，即团队会发生多次集成。每次集成都通过自动化的构建（包括代码编译，发布，自动化测试）来验证，从而尽早地发现集成错误。



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



综上所述DSGP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： DSGP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DSGP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



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DSGP平台**，更高效，更稳定，功能更强大！

对比项目	Dubbo	腾讯云DSGP
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

Spring Cloud 主流的子项目如下：

Spring Cloud Config：配置管理开发工具包，可以让你把配置放到远程服务器，目前支持本地存储、Git以及Subversion。

Spring Cloud Bus：事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化，可与Spring Cloud Config联合实现热部署。

Spring Cloud Netflix：针对多种Netflix组件提供的开发工具包，其中包括Eureka、Hystrix、Zuul、Archaius等。

Netflix Eureka：云端服务注册、发现、负载均衡，一个基于 REST 的服务，用于定位服务，以实现云端的负载均衡和中间层服务器的故障转移。

Netflix Hystrix：容错管理工具，旨在通过控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

Netflix Zuul：边缘服务工具，是提供动态路由，监控，弹性，安全等的边缘服务。类似于API GateWay

Netflix Archaius：配置管理API，包含一系列配置管理API，提供动态类型化属性、线程安全配置操作、轮询框架、回调机制等功能。

Spring Cloud for Cloud Foundry：通过Oauth2协议绑定服务到CloudFoundry，CloudFoundry是VMware推出的开源PaaS云平台。

Spring Cloud Sleuth：日志收集工具包，封装了Dapper,Zipkin和HTrace操作。

Spring Cloud Data Flow：大数据操作工具，通过命令行方式操作数据流。

Spring Cloud Security：安全工具包，为你的应用程序添加安全控制，主要是指OAuth2。

Spring Cloud Consul：封装了Consul操作，consul是一个服务发现与配置工具，与Docker容器可以无缝集成。

Spring Cloud Zookeeper：操作Zookeeper的工具包，用于使用zookeeper方式的服务注册和发现。

Spring Cloud Stream：数据流操作开发包，封装了与Redis,Rabbit、Kafka等发送接收消息。

Spring Cloud CLI：基于 Spring Boot CLI，可以让你以命令行方式快速建立云组件。

二、DSGP产品使用说明

产品使用文档主要有以下部分：

1、DSGP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、DSGP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

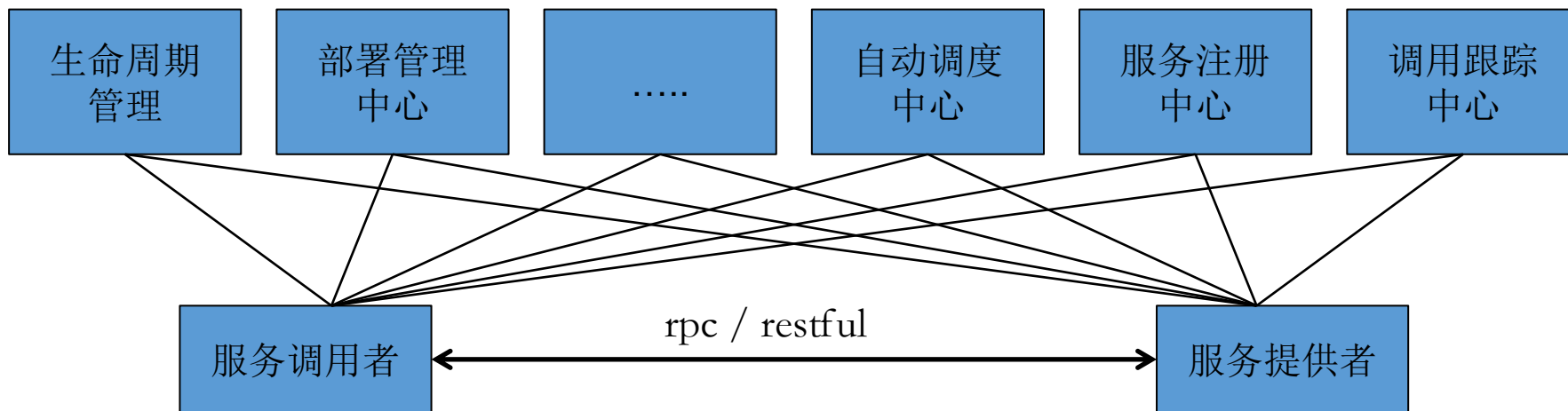
7、DSGP全局配置管理介绍

8、监控能力说明

9、产品计费，定价说明

10、Dubbo迁移到DSGP说明

11、Java 周边生态支持



步骤1: 创建DSGP实例: 购买具体的DSGP实例, 分为基础版、高级版, 支持扩容

步骤2: 常见的1个DSGP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: DSGP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

步骤4: OSS在具体的应用服务器上安装DSGP-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。DSGP-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

DSGP的控制台概览页可清晰的查看到具体服务的资源情况。

当前业务[Web层]: CDG-余额增值系统 模块: cft_yezzxt_cgi_TLinux > 登录/管理当前服务器 [旧版业务操作]

业务详情

监控告警

发布与变更

模块运维功能

登录/管理当前服务器

扩容服务器

配置环境包

配置环境脚本

配置Web服务器

在线修改业务代码

模块高级配置

访问接入

设备管理

设备自动化调度

模块 cft_yezzxt_cgi_TLinux当前服务器

扩容

刷新

对选中的服务器,您可以:

登录

下线并清理环境

删除运营

投入运营

docker镜像生成

导出设备IP

添加到CL5

从CL5删除

以 , ; 或空格分隔多个IP

IP过滤

	IP	服务器版本	机型	OS	集群	机房	机房管理单元	交换机映射Id	机架
<input type="checkbox"/>	10.209.6.86	Apache/2.2.19 (Unix)-worker	C1	TLINUX	深圳-沙河财付通通用集群	深圳电信蛇口高科DC3楼04	964	500	0304-E01
<input type="checkbox"/>	10.209.6.90	Apache/2.2.19 (Unix)-worker	C1	TLINUX	深圳-沙河财付通通用集群	深圳电信蛇口高科DC3楼04	964	500	0304-E01
<input type="checkbox"/>	10.224.134.85	Apache/2.2.19 (Unix)-worker	C1	TLINUX	深圳-沙河财付通通用集群	深圳电信蛇口高科DC2楼01	1133	643	0201-J15
<input type="checkbox"/>	10.224.137.223	Apache/2.2.19 (Unix)-worker	C1	TLINUX	深圳-沙河财付通通用集群	深圳电信蛇口高科DC2楼01	1133	481	0203-N13

DSGP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/>

maven

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



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



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>
```

Spring Boot部署demo:

(2) 编写应用的代码

```
1 package com.tencent.rancho;  
2  
3 import org.springframework.boot.*;  
4 import org.springframework.boot.autoconfigure.*;  
5 import org.springframework.stereotype.*;  
6 import org.springframework.web.bind.annotation.*;  
7  
8 @RestController  
9 @EnableAutoConfiguration  
10 public class SampleController {  
11  
12     @RequestMapping("/")  
13     String home() {  
14         return "Hello World\n";  
15     }  
16  
17     public static void main(String[] args) throws Exception {  
18         SpringApplication.run(SampleController.class, args);  
19     }  
20 }
```

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

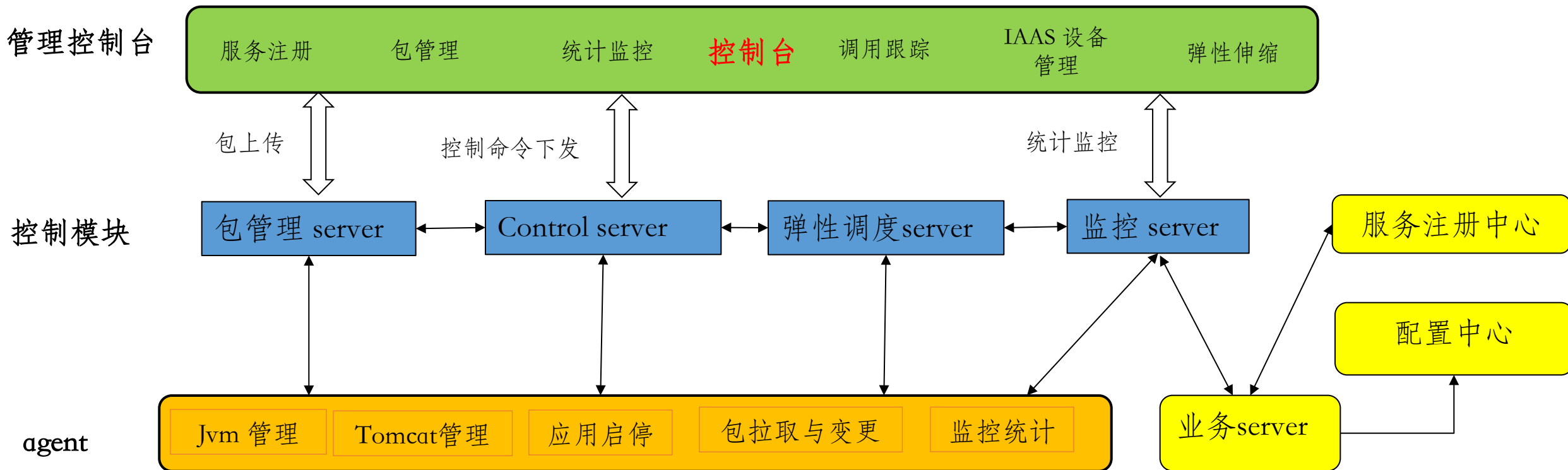
@RestController和@RequestMapping是Framework已有的注解

@EnableAutoConfiguration是Boot引入的注解

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。

DSGP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、DSGP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。



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

DSGP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DSGP平台，提供自动的代码部署服务（注意放通访问策略）

当前业务[Web层]: CDG-余额增值系统 模块: cft_yezzxt_cgi_TLinux > 配置环境包 [旧版业务操作]

关键字、域名、IP...

业务详情

监控告警

发布与变更

模块运维功能

登录/管理当前服务器

扩容服务器

配置环境包

配置环境脚本

配置Web服务器

在线修改业务代码

模块高级配置

访问接入

设备管理

设备自动化调度

任务查询

批量运维工具

cft_yezzxt_cgi_TLinux 模块:包与配置管理

+增加包

✖删除选中包

⇄调整包安装次序

搜索:

<input type="checkbox"/>	业务包	包版本	包管理员	配置最后更新	已关联配置	包关联操作
<input type="checkbox"/>	cftlogagent	v1.2.9	tsailiang;	2014-03-27 16:51:39	无配置	<div>+ C ↑ ⇄</div>
<input type="checkbox"/>	ErrorCodeAgent	v1.0.24	tsailiang	2014-04-01 15:29:24	无配置	<div>+ C ↑ ⇄</div>
<input type="checkbox"/>	L5agent	v0.0.4	terrycai	2014-07-07 14:55:57	无配置	<div>+ C ↑ ⇄</div>
<input type="checkbox"/>	get_yday_profit	v0.0.13	kansonzhang;ramonwang;tsailiang	2015-01-19 18:45:41	无配置	<div>+ C ⇄</div>
<input type="checkbox"/>	auto_scp_yday_profit	v0.0.4	kansonzhang;silonli;wenlonwang;louisjiang;jasonpeng;jollexu	2015-03-25 11:29:23	无配置	<div>+ C ⇄</div>
<input type="checkbox"/>	generate_zsjl_hs300	v0.0.3	jollexu	2015-05-07 10:33:18	无配置	<div>+ ⚠ ↑ ⇄</div>
<input type="checkbox"/>	lp_agent	v0.0.1	lustrezhang;ramonwang	2015-10-09 14:55:16	无配置	<div>+ C ⇄</div>
<input type="checkbox"/>	ntp_crontab	v0.0.1	lustrezhang	2015-12-21 18:18:29	无配置	<div>+ ⚠ ⇄</div>
<input type="checkbox"/>	oldirty_agent_fund	v0.0.1	lustrezhang	2016-04-18 17:31:08	无配置	<div>+ C ⇄</div>
<input type="checkbox"/>	log_clear_batch_fund	v0.0.2	lustrezhang	2016-06-03 15:50:38	无配置	<div>+ C ⇄</div>

从第 1 到第 10 条数据; 总有 10 条记录 每页显示 100 条记录

< 1 >

服务生命周期管理 – 服务发布



容量监控
告警配置
发布与变更
模块运维功能
登录/管理当前服务器
扩容服务器
配置环境包
配置环境脚本
配置Web服务器
在线修改业务代码
模块高级配置
访问接入
设备管理
待上线设备
已上线设备
设备自动化调度

搜索:

发布模块	类型	单号	描述	建单时间	成功数	失败数	发布人	状态/操作
gongyi_weixin_houdong	Web服务器配置	814550262	添加给市部的201799公益取捐步人次 CGI	2017-08-16 09:06:59	11	0	jayyi	✓ [查看]
NGX_950000	Web服务器配置	826017361	增加新服务	2017-09-08 01:45:13	59	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	826001593	ProjTransQueryTop.fcgi	2017-09-08 00:55:24	59	0	yijiazhang	✓ [查看]
NGX_950000	Web服务器配置	825376192	增加新服务	2017-09-06 20:08:02	59	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	824784207	增加新服务	2017-09-05 16:52:00	59	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	824228380	增加新服务	2017-09-04 14:45:22	29	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	824227035	修改服务	2017-09-04 14:41:45	0	29	kietchen	[继续发布]
NGX_950000	Web服务器配置	824223268	CftQuerySign.fcgi配置丢失	2017-09-04 14:33:18	29	0	yijiazhang	✓ [查看]
NGX_950000	Web服务器配置	823688506	增加新服务	2017-09-03 11:54:27	29	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	822651119	增加新服务	2017-09-01 10:50:38	29	0	kietchen	✓ [查看]
NGX_950000	Web服务器配置	822124119	朋友圈CGI更新	2017-08-31 11:25:58	0	0	jiahaochen	[继续发布]
NGX_950000	Web服务器配置	822120879	增加99朋友圈CGI	2017-08-31 11:18:49	29	0	jiahaochen	✓ [查看]
NGX_950000	Web服务器配置	822104262	99	2017-08-31 10:38:52	0	0	jiahaochen	[继续发布]
NGX_950000	Web服务器配置	821588474	1799	2017-08-30 11:12:32	29	0	p shuntan	✓ [查看]

当前业务[Web层] : CDG-公益微信WEB版

模块 : 请选择模块 > 业务信息

旧版业务操作

业务详情

业务信息

添加模块

业务负责人

监控告警

发布与变更

模块运维功能

访问接入

设备管理

设备自动化调度

任务查询

批量运维工具

业务备注

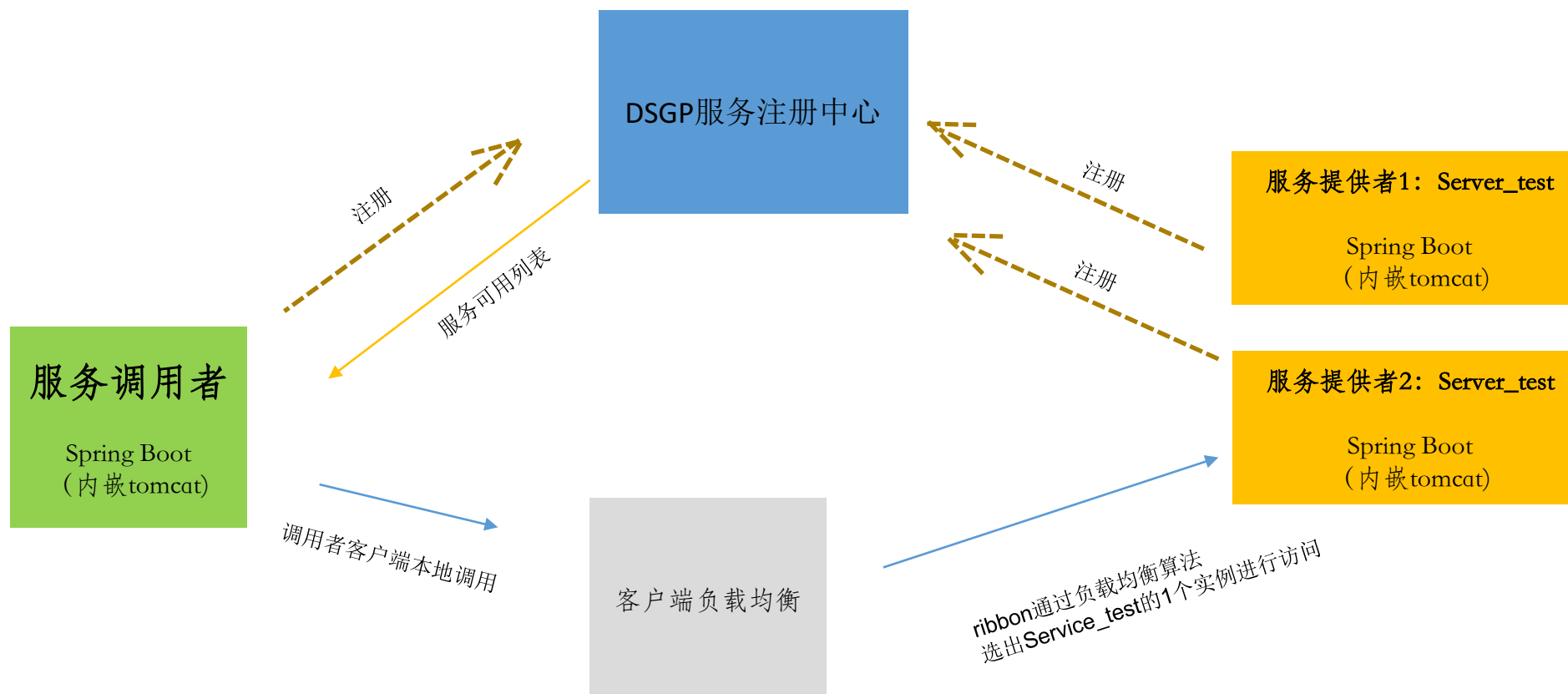
+ 添加备注

业务名称	业务ID	访问控制层	资源集群	模块数	服务器数	业务类型
CDG-公益微信WEB版	950000	有	上海财付通通用集群	10	201	普通业务

模块信息

模块名	模块ID	模块描述	自动扩容	访问控制层	状态	负责人
gongyi_weixin_99	950006	99公益日朋友圈广告专用	不允许	否	运行中	jayyi;kietchen
gongyi_weixin_gray	950008	灰度测试机器	不允许	否	运行中	hunterguo
gongyi_weixin_houdong	950005	公益微信活动	不允许	否	运行中	jayyi;kietchen
gongyi_weixin_htdocs	950004	公益微信静态资源	不允许	否	运行中	hunterguo

DSGP服务注册、发现，调用的客户使用说明



腾讯云DSGP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

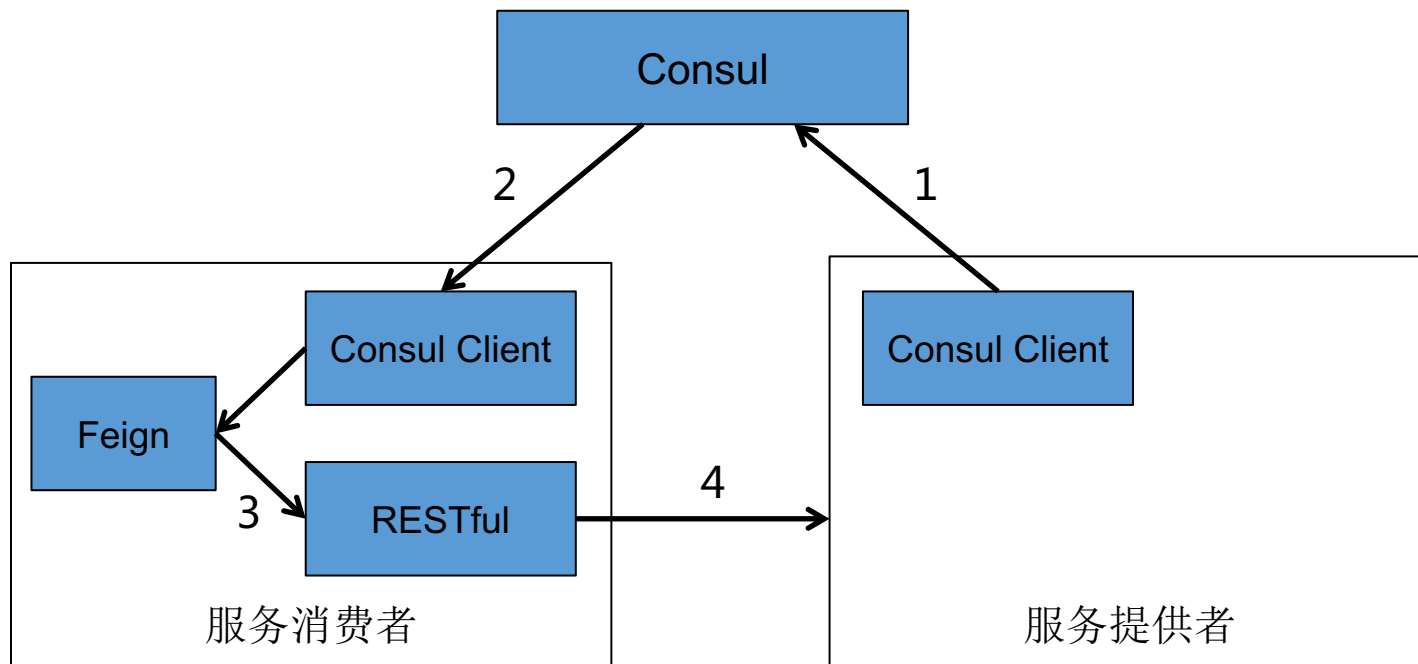
服务注册发现 + RPC调用

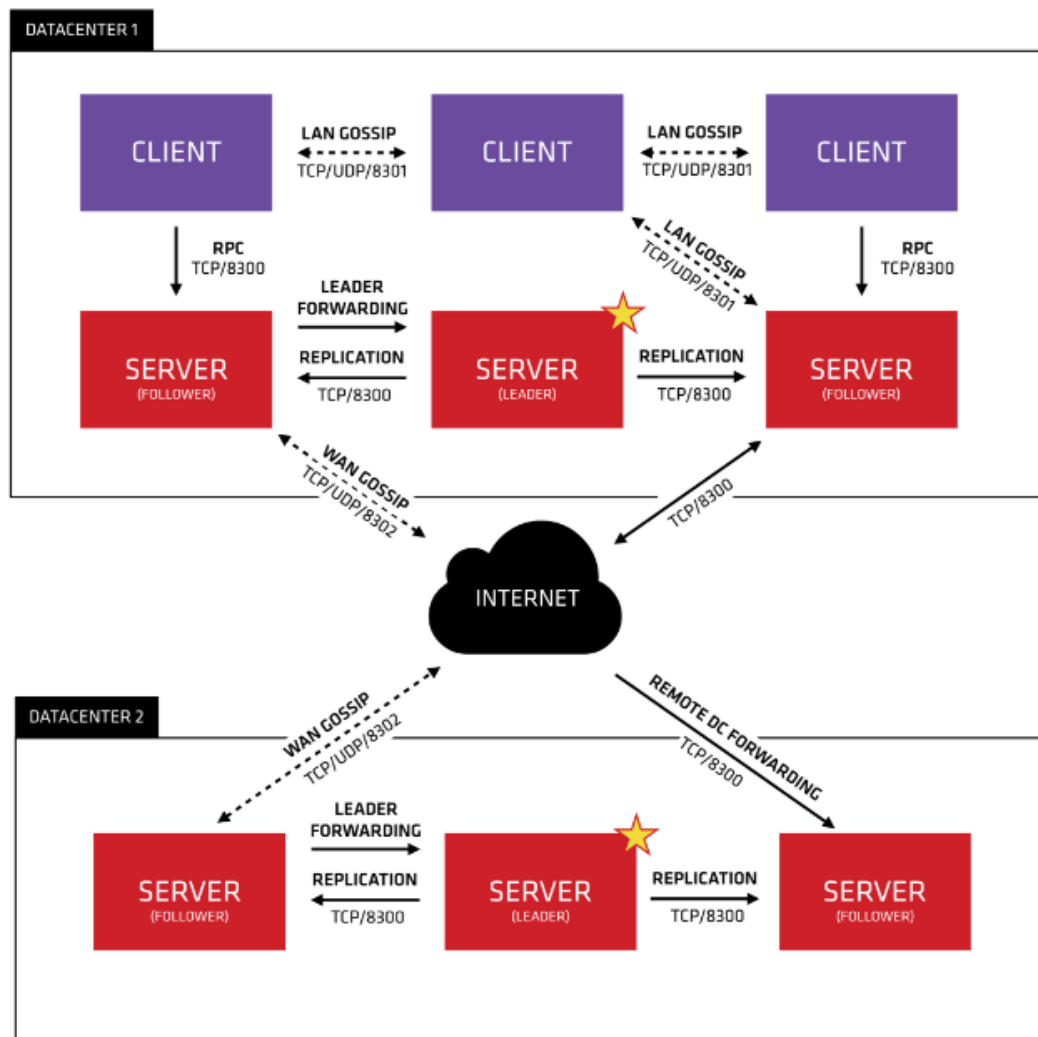
1. 服务提供者通过Consul Client向Consul发起注册

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

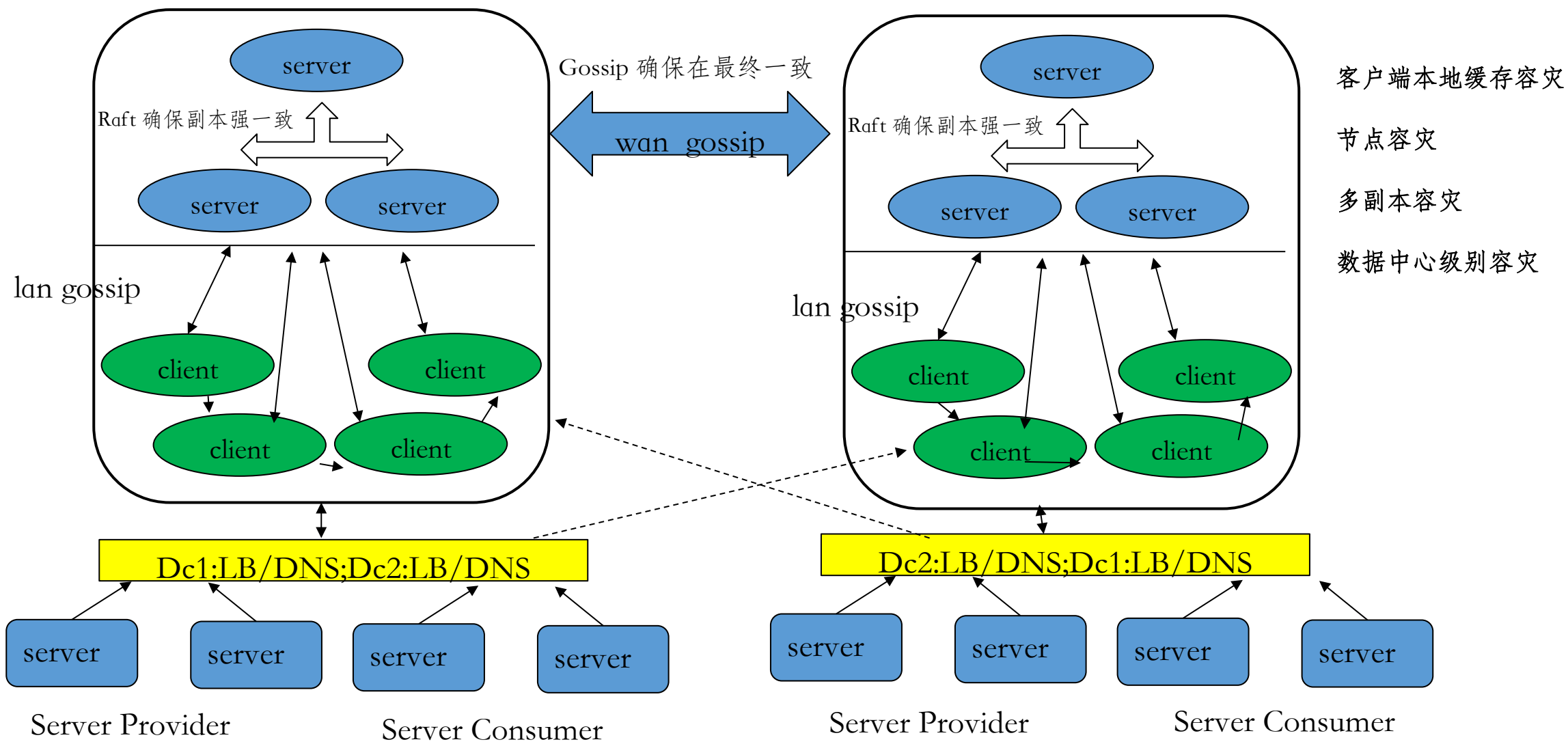




您在本地自行部署Consul服务时，一般分为Client和Server。Client接收应用的请求，将请求转发给Server，可以根据性能要求进行扩展。Server一般3或5台构成集群，负责保存服务注册信息，也可以直接接收应用的请求

腾讯云的DSGP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。





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)

SERVICES

NODES

KEY/VALUE

ACL

DC1 ▾

Filter by name

any status ▾

EXPAND

consul-client-1

1 services

consul-worker-1

1 services

consul-worker-1 10.9.10.173

SERVICES

hwapp_web

master

10.9.10.173:5000

CHECKS

Serf Health Status serfHealth

passing

NOTES

OUTPUT

Agent alive and reachable

Service 'hwapp_web' check service:hwapp_web

passing

NOTES

OUTPUT

HTTP GET http://10.9.10.173:5000: 200 OK Output: Hello World!

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

```
9      <parent>
10        <groupId>org.springframework.boot</groupId>
11        <artifactId>spring-boot-starter-parent</artifactId>
12        <version>1.5.2.RELEASE</version>
13      </parent>
14
15      <dependencies>
16        <dependency>
17          <groupId>org.springframework.cloud</groupId>
18          <artifactId>spring-cloud-starter-consul-discovery</artifactId>
19        </dependency>
20      </dependencies>
21
22      <dependencyManagement>
23        <dependencies>
24          <dependency>
25            <groupId>org.springframework.cloud</groupId>
26            <artifactId>spring-cloud-dependencies</artifactId>
27            <version>Dalston.SR2</version>
28            <type>pom</type>
29            <scope>import</scope>
30          </dependency>
31        </dependencies>
32      </dependencyManagement>
33
34      <build>
35        <plugins>
36          <plugin>
37            <groupId>org.springframework.boot</groupId>
38            <artifactId>spring-boot-maven-plugin</artifactId>
39          </plugin>
40        </plugins>
41      </build>
```

(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 }
```

(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
```

服务调用者从DSGP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

```
9      <parent>
10          <groupId>org.springframework.boot</groupId>
11          <artifactId>spring-boot-starter-parent</artifactId>
12          <version>1.5.2.RELEASE</version>
13      </parent>
14
15      <dependencies>
16          <dependency>
17              <groupId>org.springframework.cloud</groupId>
18              <artifactId>spring-cloud-starter-consul-discovery</artifactId>
19          </dependency>
20      </dependencies>
21
22      <dependencyManagement>
23          <dependencies>
24              <dependency>
25                  <groupId>org.springframework.cloud</groupId>
26                  <artifactId>spring-cloud-dependencies</artifactId>
27                  <version>Dalston.SR2</version>
28                  <type>pom</type>
29                  <scope>import</scope>
30              </dependency>
31          </dependencies>
32      </dependencyManagement>
33
34      <build>
35          <plugins>
36              <plugin>
37                  <groupId>org.springframework.boot</groupId>
38                  <artifactId>spring-boot-maven-plugin</artifactId>
39              </plugin>
40          </plugins>
41      </build>
```

(2) 编写代码

```
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 ExampleServiceTwo {
19
20     @Autowired
21     private DiscoveryClient discoveryClient;
22
23     @RequestMapping("/")
24     public String home() {
25         // 从服务注册中心获取服务Service1的访问地址列表
26         List<ServiceInstance> serviceDestination = discoveryClient.getInstances("Service1")
27
28         // 从serviceDestination中选择1个访问地址
29         // 发送请求，获取响应
30         // 其他处理
31         // return 处理结果
32     }
33
34     @RequestMapping("/health/")
35     public String health() {
36         return "[Service2-Instance1]I'm OK";
37     }
38
39     public static void main(String[] args) {
40         SpringApplication.run(ExampleServiceTwo.class, args);
41     }
42
43 }
```

(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

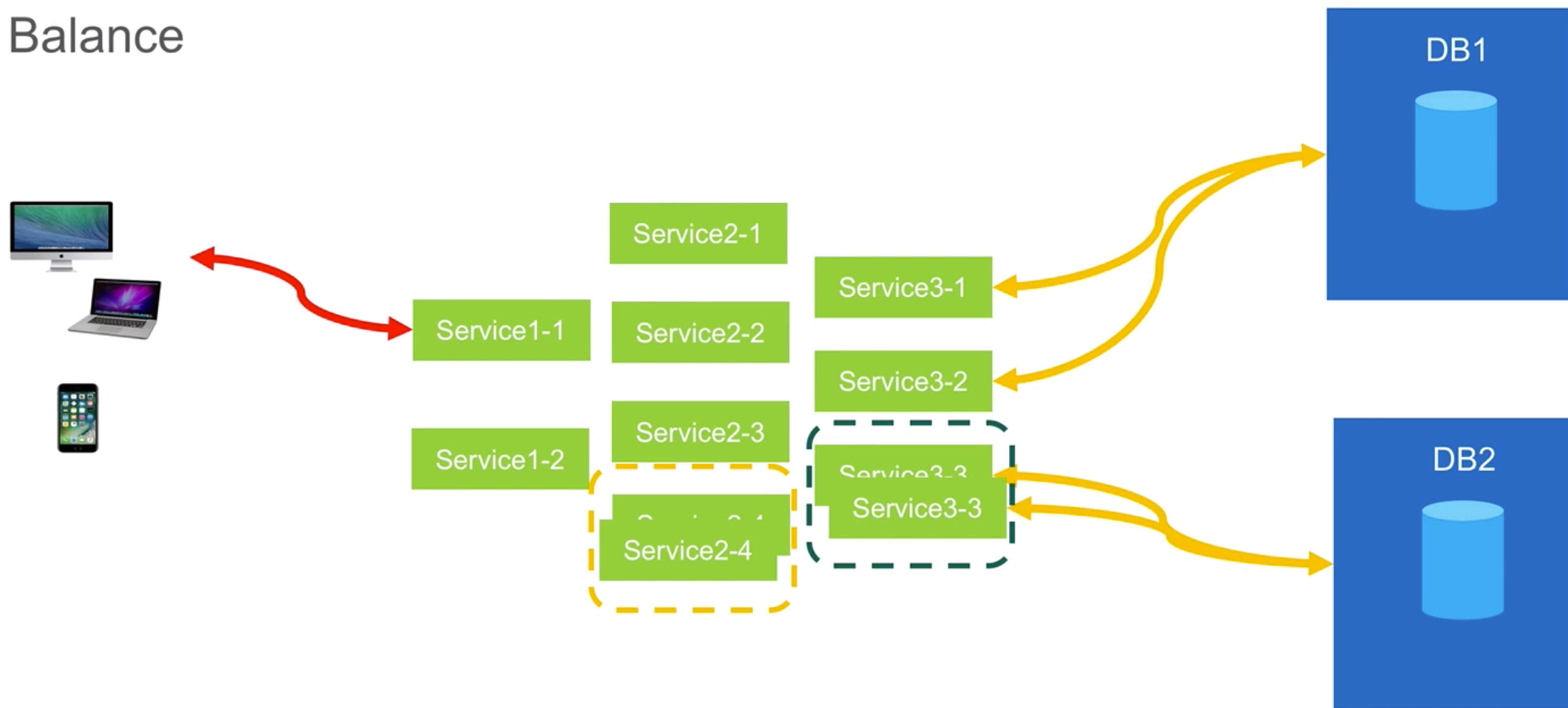
spring.cloud.consul.port=80表示服务注册中心的端口

此时该服务调用者已注册到DSGP服务注册中心

```
1 spring.application.name=Service2
2
3 server.port=8082
4
5 spring.cloud.consul.host=10.235.21.35
6 spring.cloud.consul.port=80
7
8 spring.cloud.consul.discovery.healthCheckUrl=http://10.235.20.24:8082/health/
9 spring.cloud.consul.discovery.healthCheckInterval=10s
```

DSGP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。

Load Balance



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

下面示例程序根据第一章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

```
9      <parent>
10        <groupId>org.springframework.boot</groupId>
11        <artifactId>spring-boot-starter-parent</artifactId>
12        <version>1.5.2.RELEASE</version>
13      </parent>
14
15      <dependencies>
16        <dependency>
17          <groupId>org.springframework.cloud</groupId>
18          <artifactId>spring-cloud-starter-consul-discovery</artifactId>
19        </dependency>
20        <dependency>
21          <groupId>org.springframework.cloud</groupId>
22          <artifactId>spring-cloud-starter-feign</artifactId>
23        </dependency>
24      </dependencies>
25
26      <dependencyManagement>
27        <dependencies>
28          <dependency>
29            <groupId>org.springframework.cloud</groupId>
30            <artifactId>spring-cloud-dependencies</artifactId>
31            <version>Dalston.SR2</version>
32            <type>pom</type>
33            <scope>import</scope>
34          </dependency>
35        </dependencies>
36      </dependencyManagement>
37
38      <build>
39        <plugins>
40          <plugin>
41            <groupId>org.springframework.boot</groupId>
42            <artifactId>spring-boot-maven-plugin</artifactId>
43          </plugin>
44        </plugins>
45      </build>
```

(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

```
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 import org.springframework.cloud.netflix.feign.EnableFeignClients;
13 import org.springframework.cloud.netflix.feign.FeignClient;
14
15 @SpringBootApplication
16 @RestController
17 @EnableDiscoveryClient
18 @EnableFeignClients
19 public class ExampleServiceTwo {
20
21     @Autowired
22     private DiscoveryClient discoveryClient;
23
24     @Autowired
25     private RemoteService1 service1;
26
27     @RequestMapping("/")
28     public String home() {
29
30         String info = "[Service2-Instance1]send request to Service1";
31         info += "\n[Service2-Instance1]receive response from Service1: ";
32         info += service1.call();
33
34         return info;
35     }
36 }
```

```
37 @RequestMapping("/health/")
38 public String health() {
39     return "[Service2-Instance1]I'm OK";
40 }
41
42 public static void main(String[] args) {
43     SpringApplication.run(ExampleServiceTwo.class, args);
44 }
45
46 }
47
48 // Feign将RemoteService1.call()转换为http://Service1/
49 // Feign从服务注册中心获取服务Service1的访问地址列表
50 // Feign实现客户端负载均衡
51 @FeignClient(value="Service1")
52 interface RemoteService1 {
53
54     @RequestMapping("/")
55     String call();
56
57 }
```

(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

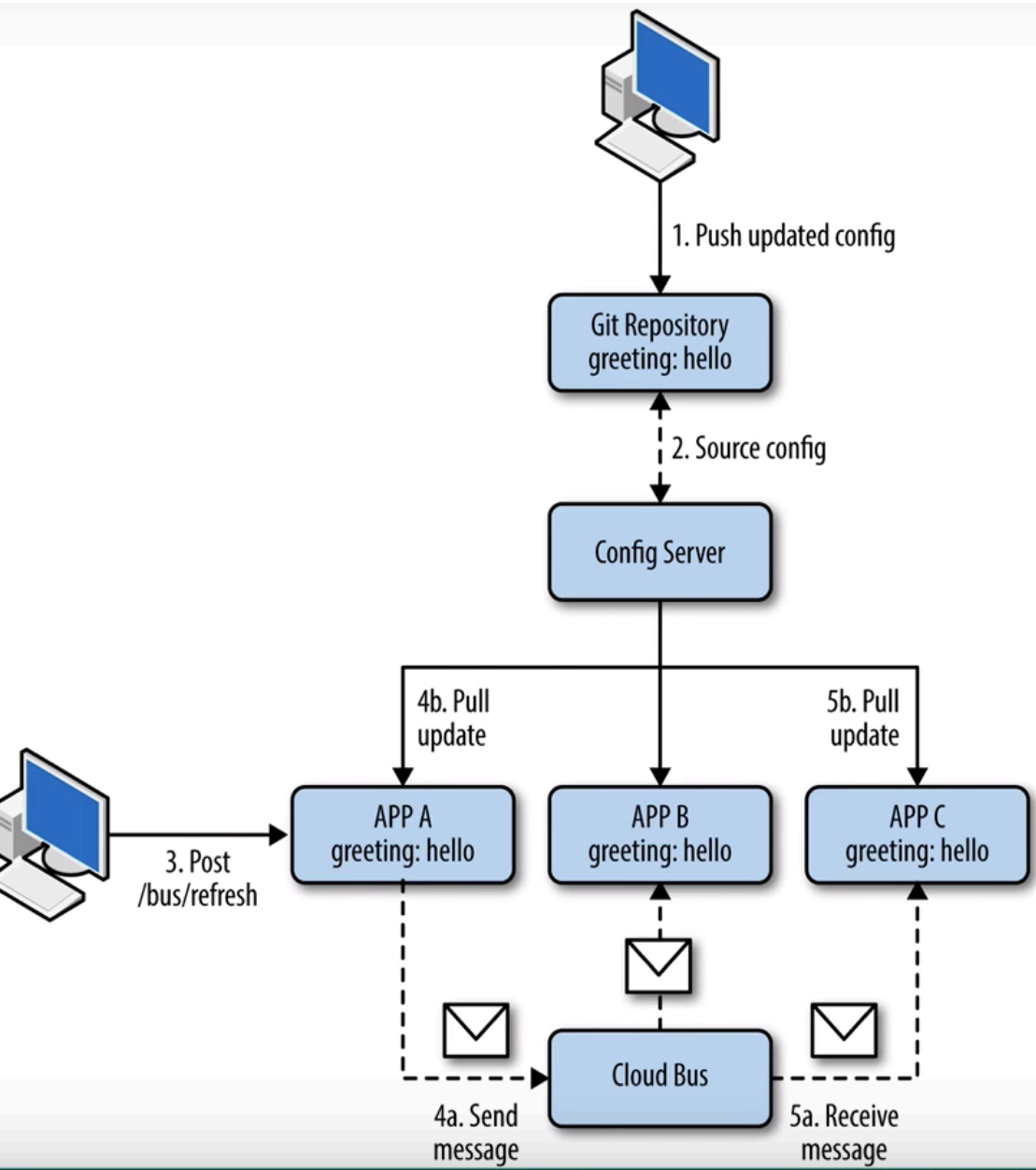
spring.cloud.consul.host=10.235.21.35表示服务注册中心的IP

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

```
1 spring.application.name=Service2
2
3 server.port=8082
4
5 spring.cloud.consul.host=10.235.21.35
6 spring.cloud.consul.port=80
7
8 spring.cloud.consul.discovery.healthCheckUrl=http://10.235.20.24:8082/health/
9 spring.cloud.consul.discovery.healthCheckInterval=10s
```

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DSGP平台结合了SpringCloud Config能力，推出了PaaS化的解决方案



DSGP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。

DSGP 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

DSGP 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。

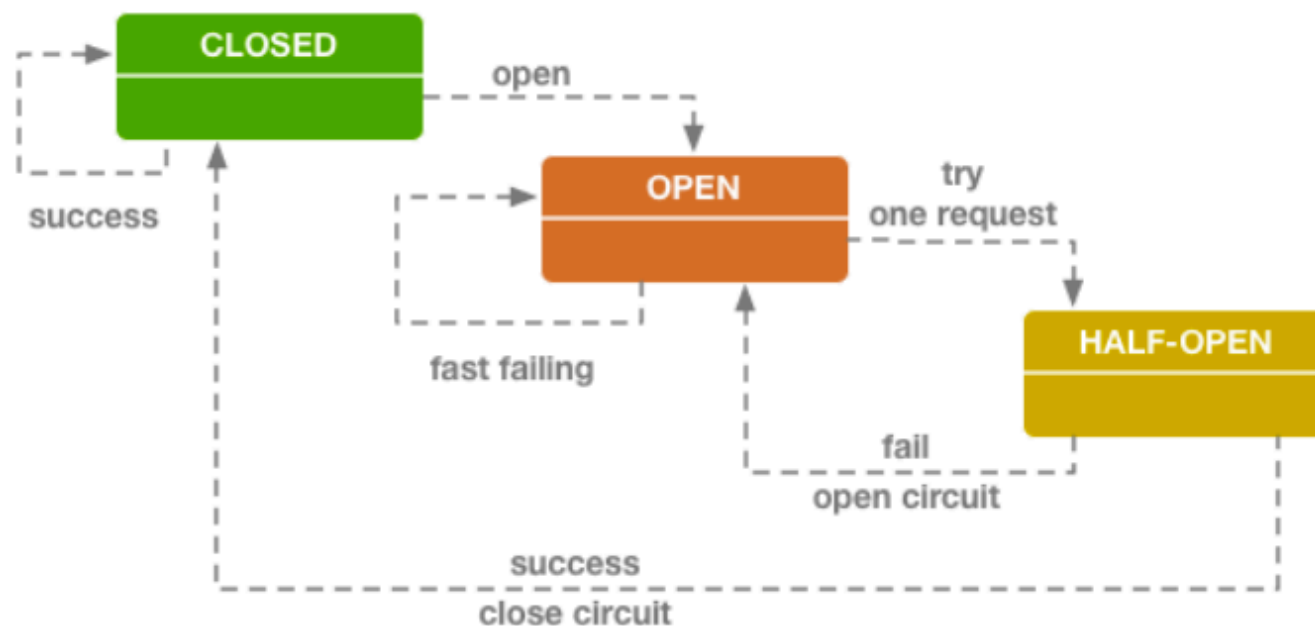
分布式系统中经常会出现某个基础服务不可用造成整个系统不可用的情况,这种现象被称为服务雪崩效应. 为了应对服务雪崩,一种常见的做法是手动服务降级. 而Hystrix的出现,给我们提供了另一种选择.

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。

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：

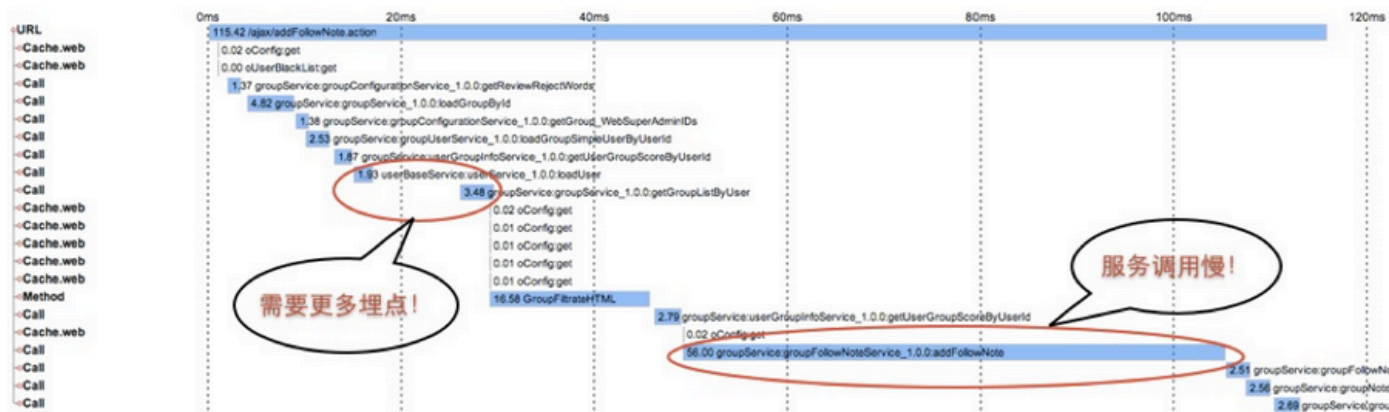
- 1、断路器统计调用每个服务的失败率
- 2、若调用某个服务的失败率超过一定阈值，则触发断路。避免有大量的同步调用的线程，阻塞于此
- 3、对于触发断路的服务请求，绝大部分不再发起远程调用，直接调用Fallback
- 4、对于触发断路的服务请求，选择少量发起远程调用，测试该服务是否恢复
- 5、若该服务恢复，则关闭断路



Circuit Breaker State Diagram

DSGP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。

1. 请求到来生成一个全局TraceID，通过TraceID可以串联起整个调用链，一个TraceID代表一次请求。
2. 除了TraceID外，还需要SpanID用于记录调用父子关系。每个服务会记录下parent id和span id，通过他们可以组织一次完整调用链的父子关系。
3. 一个没有parent id的span成为root span，可以看成调用链入口。
4. 所有这些ID可用全局唯一的64位整数表示；
5. 整个调用过程中每个请求都要透传TraceID和SpanID。
6. 每个服务将该次请求附带的TraceID和附带的SpanID作为parent id记录下，并且将自己生成的SpanID也记录下。
7. 要查看某次完整的调用则只要根据TraceID查出所有调用记录，然后通过parent id和span id组织起整个调用父子关系。



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

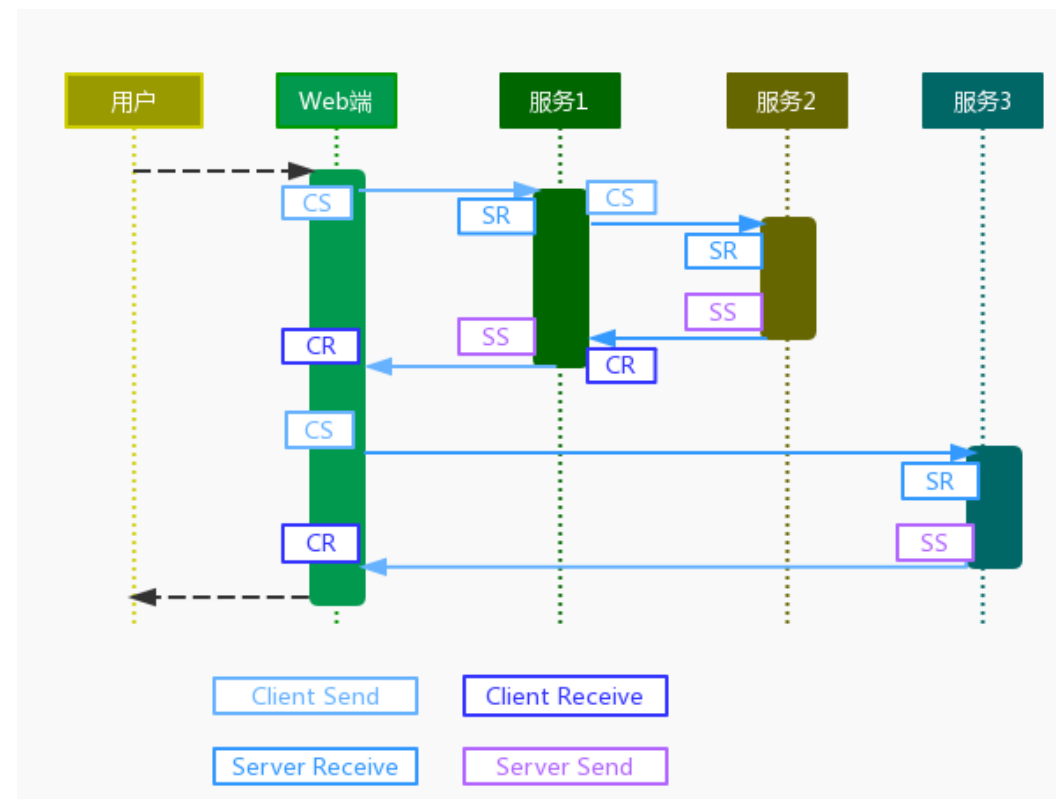
Annotation: 业务自定义的埋点信息, 可以是sql、用户ID等关键信息

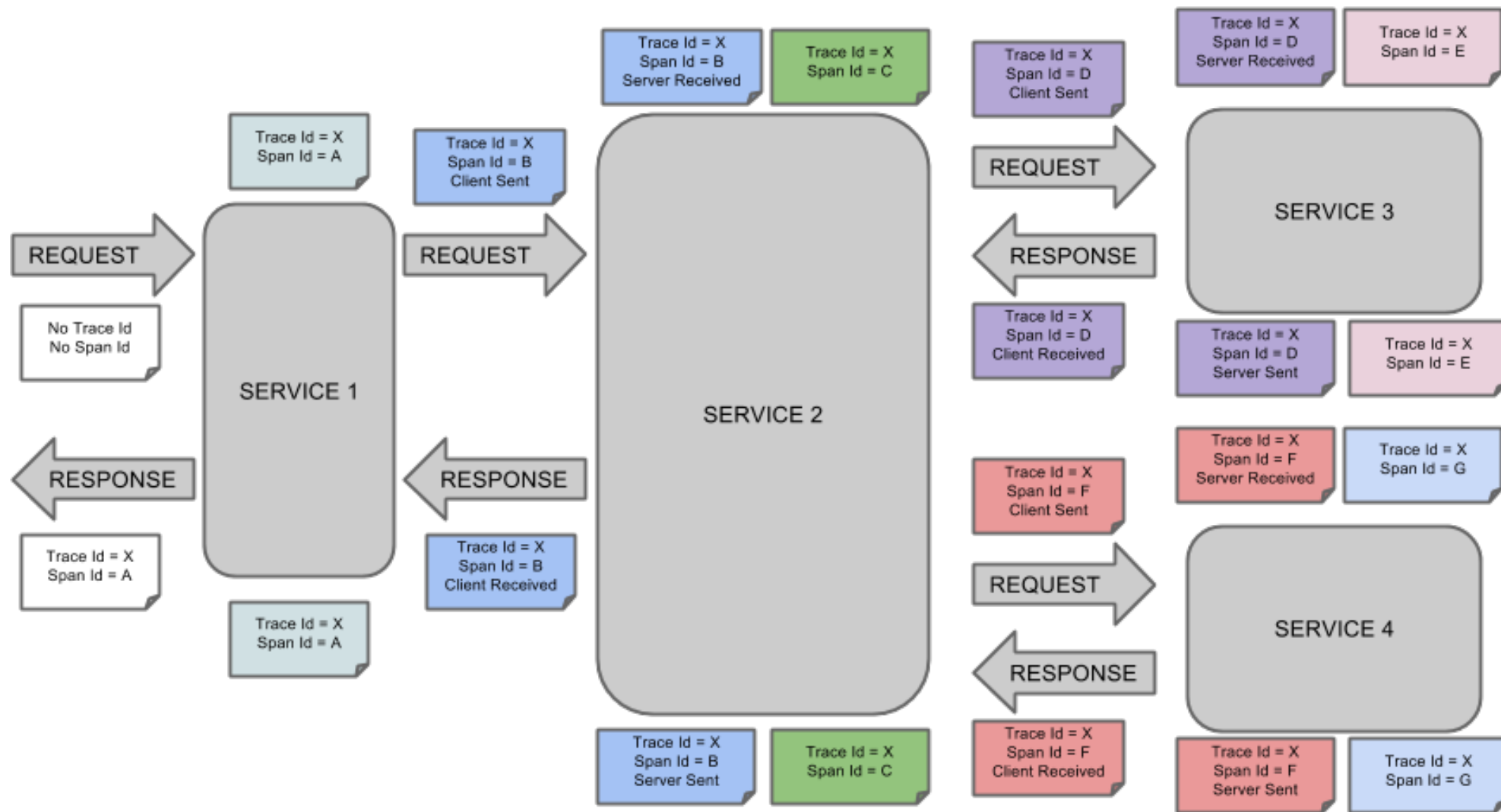
如何对业务透明, 如何在上下文中记录:

底层框架支持, 无需用户干预

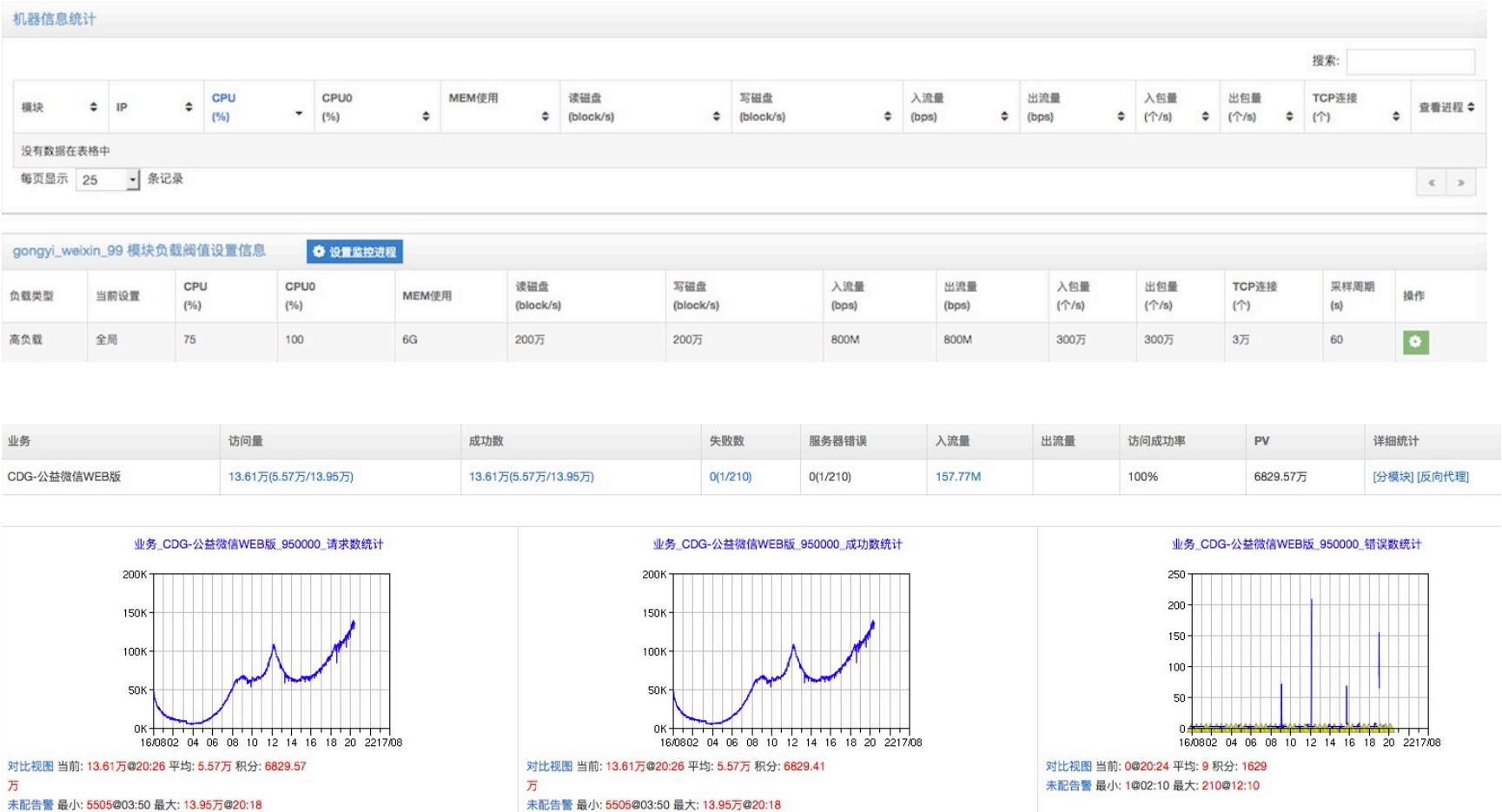
框架埋点四个阶段:

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





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



企业级分布式服务治理平台DSGP产品价格（单位：人民币元/月）

- 1、按照包年包月的方式进行付费购买，支持包年折扣（如包年付费减免2个月）
- 2、计费维度按照应用部署的节点数来计算。例如：100个应用，共运行在20台机器上，那么20台就DSGP的计费维度
- 3、企业版提供私有化部署方案，包含上云驻场工程师技术支持、集成服务、集群扩容的费用

最大支持节点数量	基础版	企业版
50	1000	1000
100	2000	20000
500	-	100000
无限制		200000

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())));
    }
}
```

腾讯云的DSGP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！**

DSGP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DSGP服务

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

Spring Main Projects提供了丰富的框架，每个项目解决一类问题：如Spring IO Platform解决依赖包的版本问题。Spring Framework提供注入的功能、访问数据库的接口、管理网络连接等；Spring MVC将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响；这三个模块基本上java的开发都会使用。

DSGP平台对Java开发者非常友好，以下框架依然适用，无改造量

Main Projects

From configuration to security, web apps to big data – whatever the infrastructure needs of your application may be, there is a Spring Project to help you build it. Start small and use just what you need – Spring is modular by design.



SPRING IO PLATFORM

Provides a cohesive, versioned platform for building modern applications. It is a modular, enterprise-grade distribution that delivers a curated set of dependencies.



SPRING BOOT

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.



SPRING FRAMEWORK

Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.

在Java开发中，多个项目模块之间的融合，很多情况下会出现Spring版本冲突 所以Spring IO platform 终于出现了。Spring IO 是可集成的，构建现代化应用的版本平台。Spring IO 是模块化的，企业级的分布式系统，包括了一系列的依赖，使得开发者仅能对自己所需的部分进行完全的部署控制。Spring IO 是 100% 开源，可靠和模块化的。主要特性如下：

- 一个平台，多个工作负载 - Web 构建，集成，批处理，响应式或者大数据应用
- 极致简化 Spring Boot 的开发经验
- 提供开箱即用的生产特性
- 模块化平台，允许开发者只部署他们需要的那部分
- 支持嵌入式运行时，传统应用服务器和 PaaS 部署
- 仅仅依赖 Java SE，支持 Groovy, Grails 和一部分 Java EE
- 可以结合现有的依赖系统运行（Maven 和 Gradle）
- 支持 JDK 7 和 8*

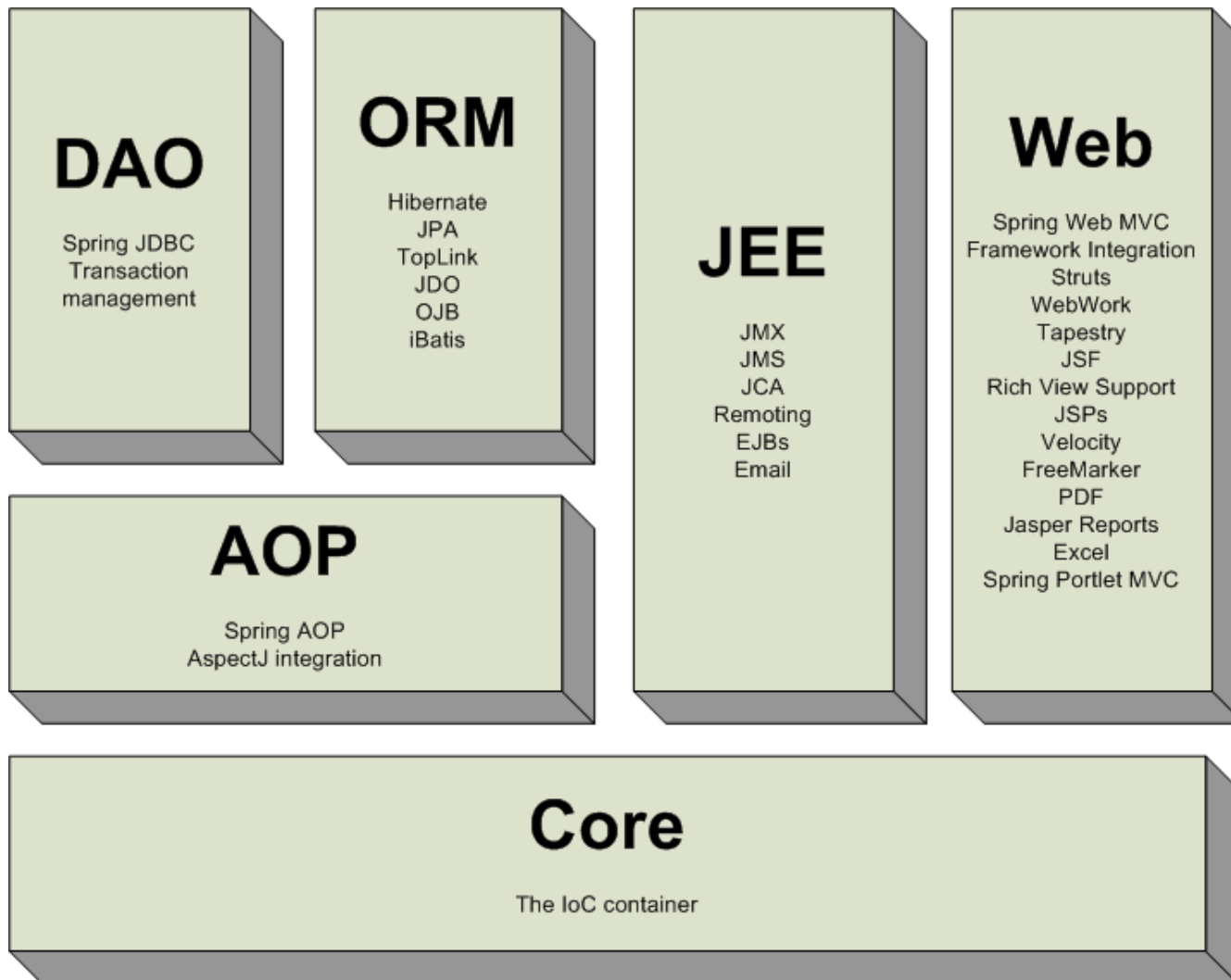
pom.xml文件里面加入依赖管理：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.spring.platform</groupId>
      <artifactId>platform-bom</artifactId>
      <version>1.1.4.RELEASE</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

然后构建你自己的应用程序时候 添加自己需要的依赖 而不用去关心Spring的版本问题：

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
</dependency>
```

Spring framework主要由7大模块组成，它们提供了企业级开发需要的所有功能，而且每个模块都可以单独使用，也可以和其他模块组合使用，灵活且方便的部署可以使开发的程序更加简洁灵活。上图所示即7个模块的部署示意图。



1) 核心模块

Spring Core模块是Spring的核心容器，它实现了IOC模式，提供了Spring框架的基础功能。此模块中包含的BeanFactory类是Spring的核心类，负责JavaBean的配置与管理。它采用Factory模式实现了IOC即依赖注入。谈到JavaBean，它是一种Java类，它遵从一定的设计模式，使它们易于和其他开发工具和组件一起使用。定义JavaBean是一种JAVA语言写成的可重用组件。要编写JavaBean，类必须是具体类和公共类，并且具有无参数的构造器。

2) Context模块

Spring Context模块继承BeanFactory（或者说Spring核心）类，并且添加了事件处理、国际化、资源装载、透明装载、以及数据校验等功能。它还提供了框架式的Bean的访问方式和很多企业级的功能，如JNDI访问、支持EJB、远程调用、集成模板框架、Email和定时任务调度等。

3) AOP模块

Spring集成了所有AOP功能。通过事务管理可以使任意Spring管理的对象AOP化。Spring提供了用标准Java语言编写的AOP框架，它的大部分内容都是基于AOP联盟的API开发的。它使应用程序抛开EJB的复杂性，但拥有传统EJB的关键功能。

4) DAO模块

DAO是Data Access Object的缩写，DAO模式思想是将业务逻辑代码与数据库交互代码分离，降低两者耦合。通过DAO模式可以使结构变得更为清晰，代码更为简洁。DAO模块提供了JDBC的抽象层，简化了数据库厂商的异常错误（不再从SQLException继承大批代码），大幅度减少代码的编写，并且提供了对声明式事务和编程式事务的支持。

5) ORM映射模块

Spring ORM 模块提供了对现有ORM框架的支持，各种流行的ORM框架已经做得非常成熟，并且拥有大规模的市场，Spring没有必要开发新的ORM工具，它对Hibernate提供了完美的整合功能，同时也支持其他ORM工具。注意这里Spring是提供各类的接口（support），目前比较流行的下层数据库封闭映射框架，如 ibatis, Hibernate等。

6) Web模块

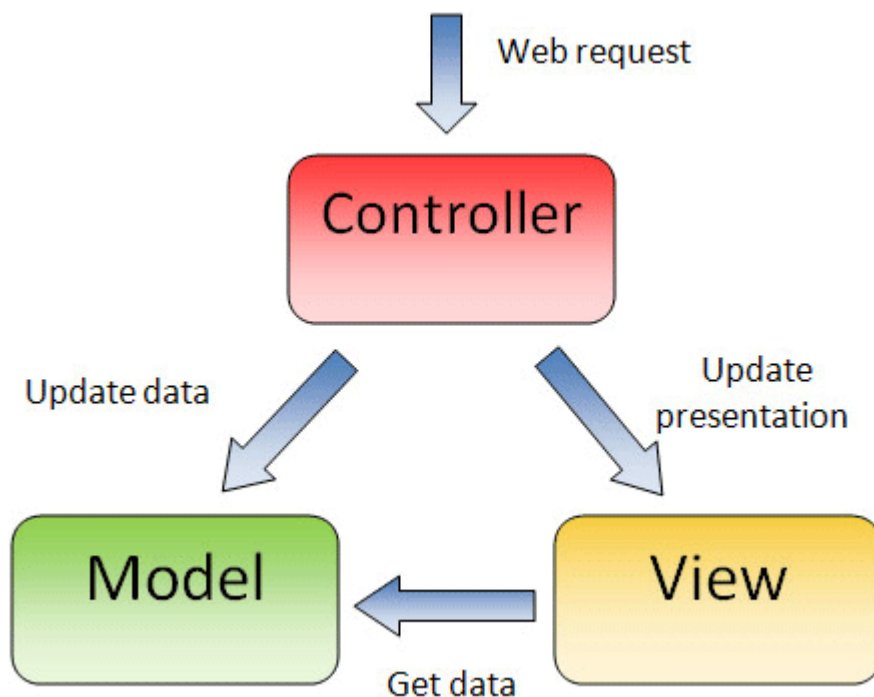
此模块建立在Spring Context 基础之上，它提供了Servlet监听器的Context 和 Web应用的上下文。对现有的Web框架，如JSF、Tapestry、Struts等，提供了集成。Struts是建立在MVC 这种公认的好的模式上的，Struts在M、V和C上都有涉及，但它主要是提供一个好的控制器和一套定制的标签库上，也就是说它的着力点在C和V上，因此，它天生就有MVC所带来的一系列优点，如：结构层次分明，高可重用性，增加了程序的健壮性和可伸缩性，便于开发与设计分工，提供集中统一的权限控制、校验、国际化、日志等等。

7) MVC模块

Spring WebMVC模块建立在Spring核心功能之上，这使它能拥有Spring框架的所有特性，能够适应多种多视图、模板技术、国际化和验证服务，实现控制逻辑和业务逻辑的清晰分离。

模型-视图-控制器（MVC）是一个众所周知的以设计界面应用程序为基础的[设计模式](#)。它主要通过分离模型、视图及控制器在应用程序中的角色将业务逻辑从界面中解耦。通常，模型负责封装应用程序数据在视图层展示。视图仅仅只是展示这些数据，不包含任何业务逻辑。

控制器负责接收来自用户的请求，并调用后台服务（manager或者dao）来处理业务逻辑。处理后，后台业务层可能会返回了一些数据在视图层展示。控制器收集这些数据及准备模型在视图层展示。MVC模式的核心思想是将业务逻辑从界面中分离出来，允许它们单独改变而不会相互影响。



Spring MVC运行流程

(1) Http请求：客户端请求提交到DispatcherServlet。

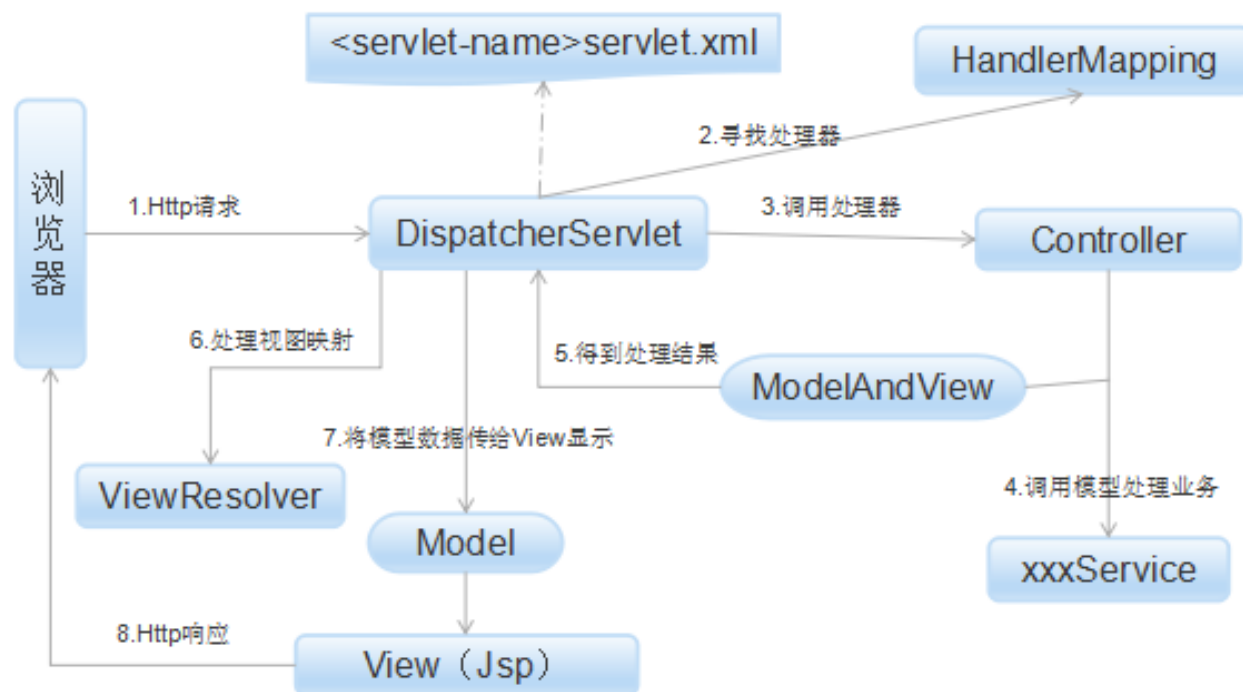
(2) 寻找处理器：由DispatcherServlet控制器查询一个或多个HandlerMapping，找到处理请求的Controller。

(3) 调用处理器：DispatcherServlet将请求提交到Controller。

(4)(5)调用业务处理和返回结果：Controller调用业务逻辑处理后，返回ModelAndView。

(6)(7)处理视图映射并返回模型：DispatcherServlet查询一个或多个ViewResolver视图解析器，找到ModelAndView指定的视图。

(8) Http响应：视图负责将结果显示到客户端。



Thank You!