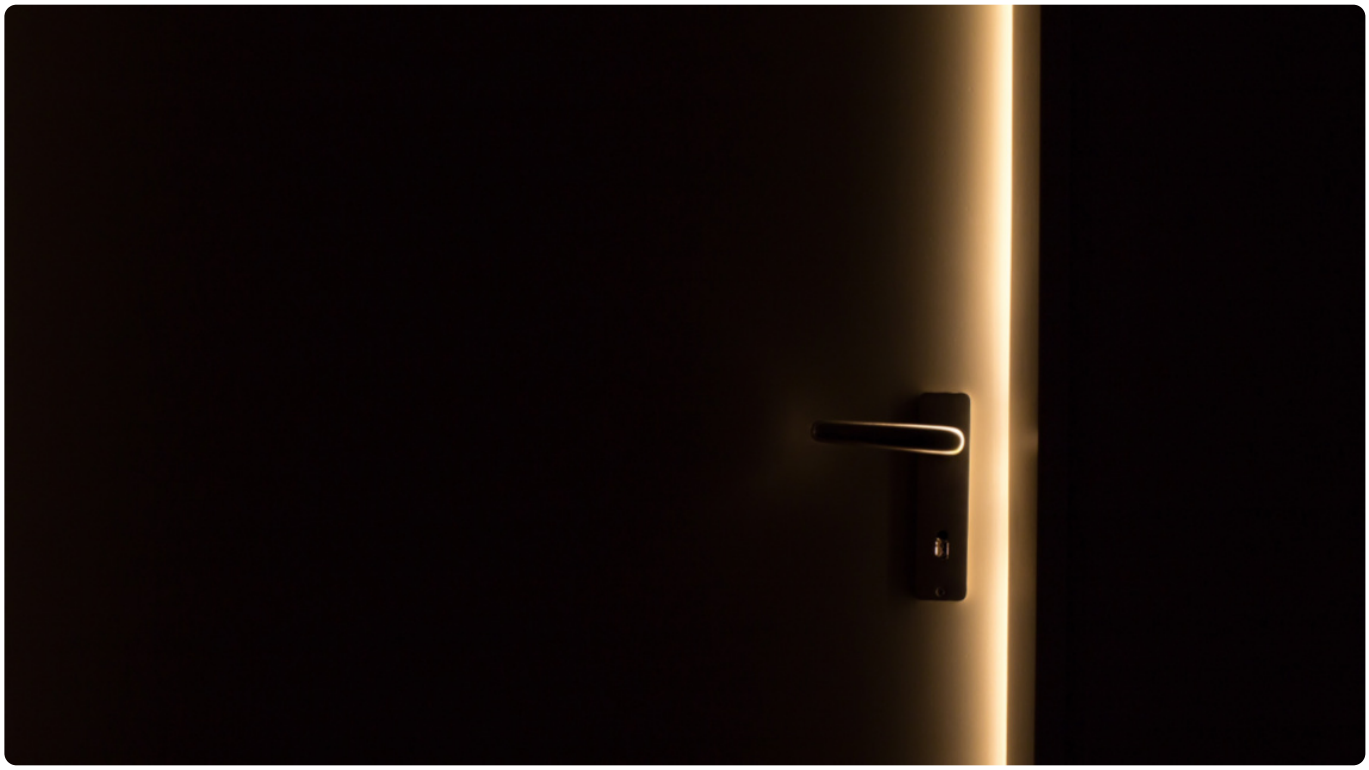
03 | 初探微服务架构

2018-08-28 胡忠想

从0开始学微服务 进入课程 >

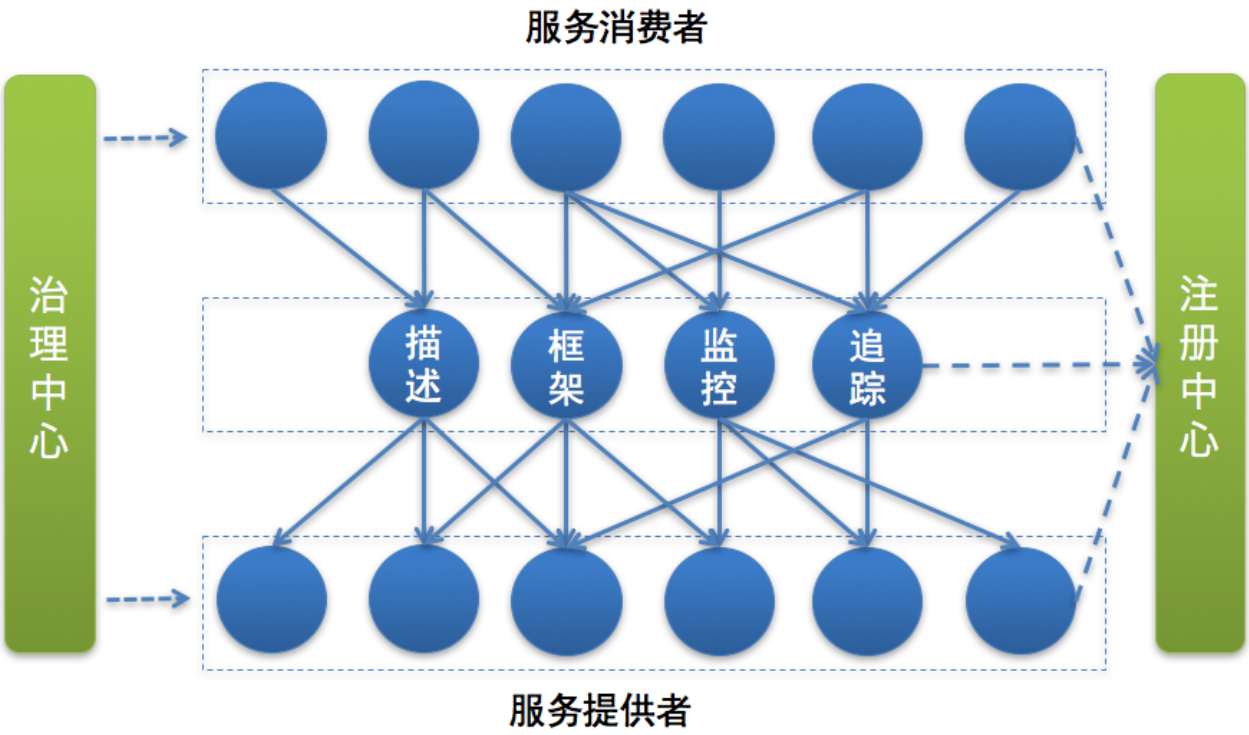


讲述：胡忠想
时长 09:31　大小 4.37M

上一期我给你讲了什么时候应该进行服务化，以及服务化拆分的两种方式即横向拆分和纵向拆分，最后还提到了引入微服务架构需要解决的问题。

我想你一定很好奇微服务架构到底是什么样子的，接下来我们一起走进微服务架构，来看看它的各个组成部分。

下面这张图是我根据自己的经验，绘制的微服务架构的模块图，在具体介绍之前先来看下一次正常的服务调用的流程。



首先服务提供者（就是提供服务的一方）按照一定格式的服务描述，向注册中心注册服务，声明自己能够提供哪些服务以及服务的地址是什么，完成服务发布。

接下来服务消费者（就是调用服务的一方）请求注册中心，查询所需要调用服务的地址，然后以约定的通信协议向服务提供者发起请求，得到请求结果后再按照约定的协议解析结果。

而且在服务的调用过程中，服务的请求耗时、调用量以及成功率等指标都会被记录下来用作监控，调用经过的链路信息会被记录下来，用于故障定位和问题追踪。在这期间，如果调用失败，可以通过重试等服务治理手段来保证成功率。

总结一下，微服务架构下，服务调用主要依赖下面几个基本组件：

服务描述

注册中心

服务框架

服务监控

服务追踪

服务治理

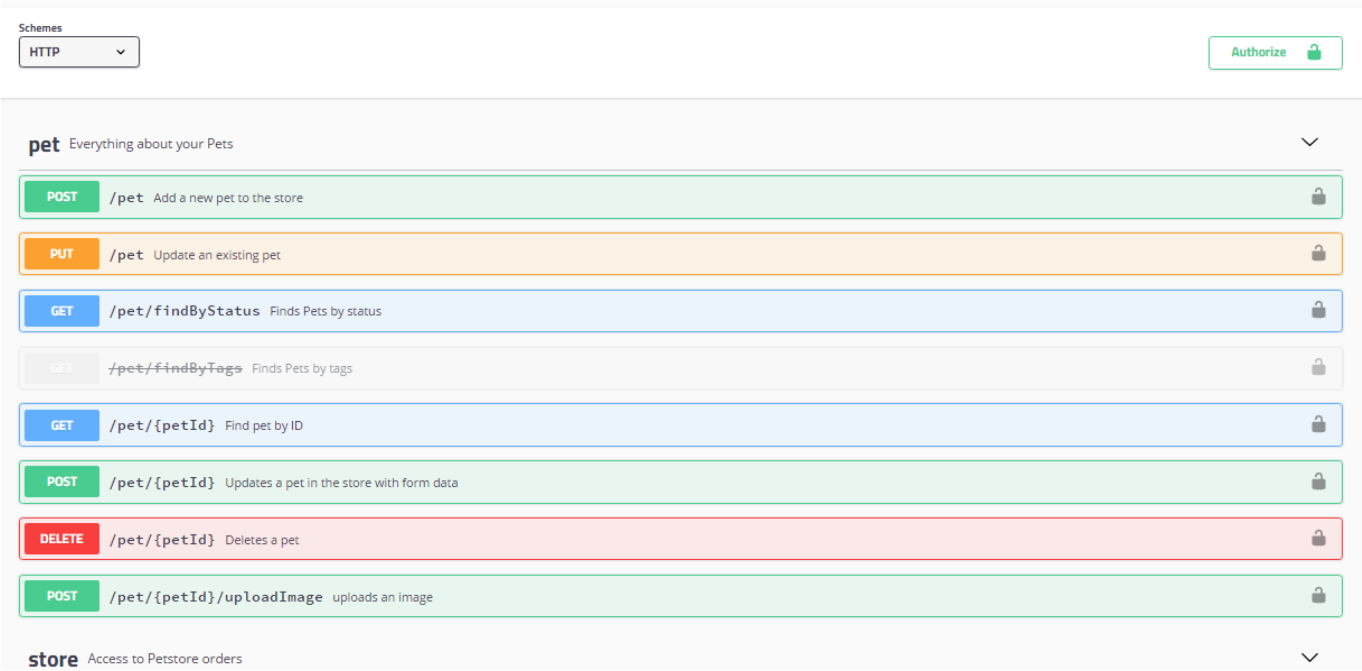
接下来，我来为你一一介绍这些组件。

服务描述

服务调用首先要解决的问题就是服务如何对外描述。比如，你对外提供了一个服务，那么这个服务的服务名叫什么？调用这个服务需要提供哪些信息？调用这个服务返回的结果是什么格式的？该如何解析？这些就是服务描述要解决的问题。

常用的服务描述方式包括 RESTful API、XML 配置以及 IDL 文件三种。

其中，RESTful API 方式通常用于 HTTP 协议的服务描述，并且常用 Wiki 或者Swagger来进行管理。下面是一个 RESTful API 方式的服务描述的例子。



XML 配置方式多用作 RPC 协议的服务描述，通过 *.xml 配置文件来定义接口名、参数以及返回值类型等。下面是一个 XML 配置方式的服务描述的例子。

```
<!-- 导出接口 -->
<motan:service ref="sinaUserService" requestTimeout="50" retries="2"
    interface="cn.sina.api.data.service.SinaUserService" basicService="serviceBasicConfig" />
<motan:service ref="sinaUserMappingService" requestTimeout="50" retries="2"
    interface="cn.sina.api.user.service.SinaUserMappingService" basicService="serviceBasicConfig" />
<motan:service ref="userService" requestTimeout="50" retries="2"
    interface="cn.sina.api.user.service.UserService" basicService="serviceBasicConfig" />
<motan:service ref="userMappingService"  requestTimeout="50" retries="2"
    interface="cn.sina.api.user.service.UserMappingService" basicService="serviceBasicConfig" />
<motan:service ref="userEduService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.UserEduService"  basicService="serviceBasicConfig" />
<motan:service ref="careerService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.CareerService"  basicService="serviceBasicConfig" />
<motan:service ref="newTagService" requestTimeout="300" retries="1"
    interface="cn.sina.api.user.service.TagService"  basicService="serviceBasicConfig" />
<motan:service ref="userCheckService" requestTimeout="600" retries="0"
    interface="cn.sina.api.user.service.UserCheckService" basicService="serviceBasicConfig" />
<motan:service ref="userStatService" requestTimeout="600" retries="0"
    interface="cn.sina.api.user.service.UserStatService" basicService="serviceBasicConfig" />
<motan:service ref="userActivityPrivacyService" requestTimeout="50" retries="2"
    interface="cn.sina.api.user.service.UserActivityPrivacyService" basicService="serviceBasicConfig"/>
<motan:service ref="accountShipService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.AccountShipService" basicService="serviceBasicConfig" />
<motan:service ref="deliverAddressService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.DeliverAddressService" basicService="serviceBasicConfig" />
<motan:service ref="userVerifiedService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.UserVerifiedService" basicService="serviceBasicConfig" />
<motan:service ref="microContactsTaskService" requestTimeout="300" retries="0"
    interface="cn.sina.api.user.service.MicroContactsTaskService" basicService="serviceBasicConfig" />
```

IDL 文件方式通常用作 Thrift 和 gRPC 这类跨语言服务调用框架中，比如 gRPC 就是通过 Protobuf 文件来定义服务的接口名、参数以及返回值的数据结构，示例如下：

```
service HelloService {
  rpc SayHello (HelloRequest) returns (HelloRes
ponse);
}

message HelloRequest {
  string greeting = 1;
}

message HelloResponse {
  string reply = 1;
}
```

注册中心

有了服务的接口描述，下一步要解决的问题就是服务的发布和订阅，就是说你提供了一个服务，如何让外部想调用你的服务的人知道。这个时候就需要一个类似注册中心的角色，服务

提供者将自己提供的服务以及地址登记到注册中心，服务消费者则从注册中心查询所需要调用的服务的地址，然后发起请求。

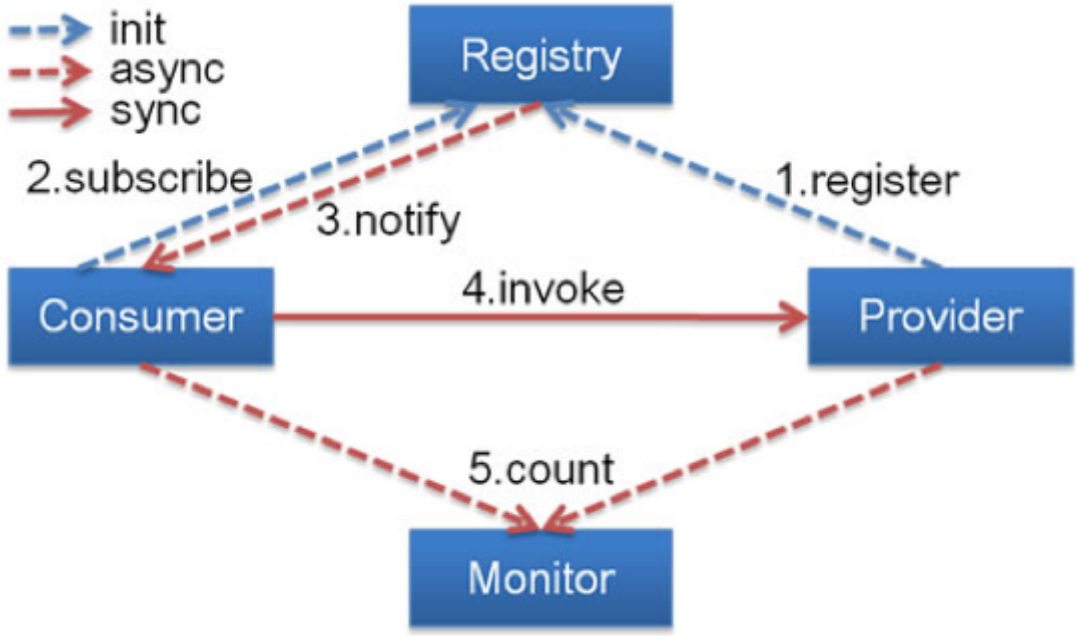
一般来讲，注册中心的工作流程是：

服务提供者在启动时，根据服务发布文件中配置的发布信息向注册中心注册自己的服务。

服务消费者在启动时，根据消费者配置文件中配置的服务信息向注册中心订阅自己所需要的服务。

注册中心返回服务提供者地址列表给服务消费者。

当服务提供者发生变化，比如有节点新增或者销毁，注册中心将变更通知给服务消费者。



服务框架

通过注册中心，服务消费者就可以获取到服务提供者的地址，有了地址后就可以发起调用。但在发起调用之前你还需要解决以下几个问题。

服务通信采用什么协议？就是说服务提供者和服务消费者之间以什么样的协议进行网络通信，是采用四层 TCP、UDP 协议，还是采用七层 HTTP 协议，还是采用其他协议？

数据传输采用什么方式？就是说服务提供者和服务消费者之间的数据传输采用哪种方式，是同步还是异步，是在单连接上传输，还是多路复用。

数据压缩采用什么格式？通常数据传输都会对数据进行压缩，来减少网络传输的数据量，从而减少带宽消耗和网络传输时间，比如常见的 JSON 序列化、Java 对象序列化以及

Protobuf 序列化等。

服务监控

一旦服务消费者与服务提供者之间能够正常发起服务调用，你就需要对调用情况进行监控，以了解服务是否正常。通常来讲，服务监控主要包括三个流程。

指标收集。就是要把每一次服务调用的请求耗时以及成功与否收集起来，并上传到集中的数据处理中心。

数据处理。有了每次调用的请求耗时以及成功与否等信息，就可以计算每秒服务请求量、平均耗时以及成功率等指标。

数据展示。数据收集起来，经过处理之后，还需要以友好的方式对外展示，才能发挥价值。通常都是将数据展示在 Dashboard 面板上，并且每隔 10s 等间隔自动刷新，用作业务监控和报警等。

服务追踪

除了需要对服务调用情况进行监控之外，你还需要记录服务调用经过的每一层链路，以便进行问题追踪和故障定位。

服务追踪的工作原理大致如下：

服务消费者发起调用前，会在本地按照一定的规则生成一个 requestid，发起调用时，将 requestid 当作请求参数的一部分，传递给服务提供者。

服务提供者接收到请求后，记录下这次请求的 requestid，然后处理请求。如果服务提供者继续请求其他服务，会在本地再生成一个自己的 requestid，然后把这两个 requestid 都当作请求参数继续往下传递。

以此类推，通过这种层层往下传递的方式，一次请求，无论最后依赖多少次服务调用、经过多少服务节点，都可以通过最开始生成的 requestid 串联所有节点，从而达到服务追踪的目的。

服务治理

服务监控能够发现问题，服务追踪能够定位问题所在，而解决问题就得靠服务治理了。服务治理就是通过一系列的手段来保证在各种意外情况下，服务调用仍然能够正常进行。

在生产环境中，你应该经常会遇到下面几种状况。

单机故障。通常遇到单机故障，都是靠运维发现并重启服务或者从线上摘除故障节点。然而集群的规模越大，越是容易遇到单机故障，在机器规模超过一百台以上时，靠传统的人肉运维显然难以应对。而服务治理可以通过一定的策略，自动摘除故障节点，不需要人为干预，就能保证单机故障不会影响业务。

单 IDC 故障。你应该经常听说某某 App，因为施工挖断光缆导致大批量用户无法使用的严重故障。而服务治理可以通过自动切换故障 IDC 的流量到其他正常 IDC，可以避免因为单 IDC 故障引起的大批量业务受影响。

依赖服务不可用。比如你的服务依赖依赖了另一个服务，当另一个服务出现问题时，会拖慢甚至拖垮你的服务。而服务治理可以通过熔断，在依赖服务异常的情况下，一段时期内停止发起调用而直接返回。这样一方面保证了服务消费者能够不被拖垮，另一方面也给服务提供者减少压力，使其能够尽快恢复。

上面是三种最常见的需要引入服务治理的场景，当然还有一些其他服务治理的手段比如自动扩缩容，可以用来解决服务的容量问题。

总结

通过前面的讲解，相信你已经对微服务架构有了基本的认识，对微服务架构的基本组件也有了初步了解。

这几个基本组件共同组成了微服务架构，在生产环境下缺一不可，所以在引入微服务架构之前，你的团队必须掌握这些基本组件的原理并具备相应的开发能力。实现方式上，可以引入开源方案；如果有充足的资深技术人员，也可以选择自行研发微服务架构的每个组件。但对于大部分中小团队来说，我认为采用开源实现方案是一个更明智的选择，一方面你可以节省相关技术人员的投入从而更专注于业务，另一方面也可以少走弯路少踩坑。不管你是采用开源方案还是自行研发，**都必须吃透每个组件的工作原理并能在此基础上进行二次开发。**

专栏后面的内容，我会带你对这几个微服务架构的基本组件进行详细剖析，让你能知其然也知其所以然。

思考题

最后你可以思考一下，微服务架构下的基本组件所解决的问题，对应到单体应用时是否存在？如果存在，解决方案是否相同？

欢迎你在留言区写下自己的思考，与我一起讨论。

上一篇 02 | 从单体应用走向服务化

下一篇 04 | 如何发布和引用服务？

精选留言 (34)

写留言

michael

2018-08-28

39

老师，服务追踪中，为什么各服务层需要生产自己的requestid，只使用客户端生成的requestid在各层传递不行吗？

展开

oddrock
2018-08-28

36

微服务架构下要解决的服务描述、服务注册与发现、服务框架、服务监控、服务跟踪、服务治理其实在soa下基本都同样存在。
soa下服务描述用wsdl，服务注册与发现用uddl，服务框架、服务监控和服务跟踪、治理基本都依赖esb，治理还要部分依赖负载均衡。
同样情况在单体简单的多，服务描述就是api，服务注册与发现就是引用jar，监控、跟踪...
展开

作者回复: 看来对soa很了解

XuToTo
2018-08-28

10

看到一位同学讨论到 http 和 rpc，我的理解是，其实 http 服务从某种程度上来说也算 rpc,之所以不使用 http 来做内部 rpc，我想有一部分原因是 http 包含了冗余的信息，并不适用于内部高效的 rpc，像是 gRPC 这样的都会利用 protocol buffer 来最大限度减少序列化后传输信息的大小。
展开

5ispy
2018-08-28

10

阿忠老师终于提到自动扩容，后面会细讲吗？这是不是支持8对明星并发出轨的关键□

作者回复: 后面有一节会讲混合云下的微服务架构

王宏达达达
2018-08-29

9

老师，微服务和分布式的区别到底在哪里呢？
展开

vick

4

2018-09-01

服务故障问题在传统架构中的解决方式，一般是远程调用时要做超时处理，很多http client的第三方包都提供timeout的处理。同理微服务架构下的熔断机制相比之下有哪些优势，期待后面教程的分析。

努力的小斌
2018-08-30

4

问requestID，大家可以看一下Google Dapper那篇论文

zhihai.t...
2019-04-26

3

微服务架构6大组件：
1、服务描述：RestfulApi (http)、xml (rpc)、IDL (grpc)
2、注册中心：注册（服务提供者->注册中心）、订阅（服务消费者->注册中心）、返回（注册中心->服务消费者）、通知（注册中心->服务消费者）
3、服务框架...

展开

大光头
2018-08-30

3

微服务下的问题在单体应用会有，但不是问题，因为天生满足。分布式之后，
接口调用，就需要服务发现，服务注册和服务间调用。
接口错误需要容错处理和熔断处理。
接口性能，需要靠扩容。
接口调用情况需要监控。...

展开

Final不基
2018-11-13

2

我们用spring cloud微服务框架。zipkin做链路监控，有办法链路到jdbc 数据库层么？

作者回复: zipkin不支持，skywalking可以

猿一代
2018-10-16

2

c502和1999992，老师，今天微博热搜赵丽颖和冯绍峰结婚的消息报出来的分别是什么原因呀？

Halo_浅色...
2018-08-29

2

为啥要两个requestId，我们以前就只用了一个来进行链路追踪，只用一个有啥缺陷吗

作者回复: 一个requestid，一个是spanid，这里是为了描述原理抽象表达这个意思

greatcl
2018-08-28

2

如果服务提供者继续请求其他服务，会在本地再生成一个自己的requestid，然后把这两个requestid都当作请求参数继续往下传递。
==========
老师你好，想问下为什么要两个requestid都往下传递呢，只用第一个requestid不是就可以追踪到了吗？

展开

作者回复: 这里没有详细描述 其实是spanid 这里是抽象描述原理

贾洵
2018-11-05

1

既然设置一个跟踪ID，为什么不设置一个uuid做为调用唯一ID。这样既可以全程跟踪，又节省寻找串联请求ID的时间？

展开

作者回复: uuid没法区分调用的层级关系

70
2018-08-28

1

服务描述，注册中心，服务框架，服务监控，服务追踪，服务治理。这些在单体服务中也存在，服务描述在任何时候都需要。注册中心和服务框架在单体服务中可能没有那么明显，单体服务中多数采用人为沟通和文档进行维护。服务监控和服务追踪，服务治理，这些部分的存在，可以让我们实时监控服务的运行状态，及时发现并解决问题，单体和微服务都是无法缺失的。在面对服务容量问题时，单体服务解决起来更加麻烦，由于没有服...

展开

eason2017

2018-08-28

1

在复杂调用中，应该还有熔断，这样避免底层服务拖死自身服务，对吧～

作者回复: 嗯，熔断是一种非常有效的服务治理手段

eason2017

2018-08-28

1

看到dubbo的影子了，嘎嘎

展开

feimeng053...

2018-08-28

1

单体应用的集群方式和微服务的治理，单体故障转移，

展开

godtrue

2019-05-17

阅后感

个人感觉理解微服务的关键环节在于理解网络通信，最简单的微服务，比如：只有一个服务提供者也只有一个服务消费者，只要他们能通信相互理解就行。

深入了理解他们通信的原理，其他的都是在机器增多，考虑网络环境的复杂多变，提供负载均衡、高可用、高性能、易维护、易定位问题、支持水平扩展的外围服务，是分布式...

展开

zhihai.t...

2019-04-26

负载均衡F5连了6台应用服务器，其中1台挂了，那么做健康巡检的时候没通过，则流量就转移到其它5台上，等待挂掉的那台进行修复。

展开