54 | 管理设计篇之“边车模式”

2018-4-5 陈皓

所谓的边车模式，对应于我们生活中熟知的边三轮摩托车。也就是说，我们可以通过给一个摩托车加上一个边车的方式来扩展现有的服务和功能。这样可以很容易地做到"控制"和"逻辑"的分离。

也就是说，我们不需要在服务中实现控制面上的东西，如监视、日志记录、限流、熔断、服务注册、协议适配转换等这些属于控制面上的东西，而只需要专注地做好和业务逻辑相关的代码，然后，由“边车”来实现这些与业务逻辑没有关系的控制功能。

边车模式设计

具体来说，你可以理解为，边车就有点像一个服务的Agent，这个服务所有对外的进出通讯都通过这个Agent来完成。这样，我们就可以在这个Agent上做很多文章了。但是，我们需要保证的是，这个Agent要和应用程序一起创建，一起停用。

边车模式有时候也叫搭档模式，或是伴侣模式，或是跟班模式。就像我们在《编程范式游记》中看到的那样，**编程的本质就是将控制和逻辑分离和解耦，而边车模式也是异曲同工**，同样是让我们在分布式架构中做到逻辑和控制分离。

对于监视、日志、限流、熔断、服务注册、协议转换等等这些功能，其实都是大同小异，甚至是完全可以做成标准化的组件和模块的。一般来说，我们有两种方式。

一种是通过SDK、Lib或Framework软件包方式，在开发时与真实的应用服务集成起来。

另一种是通过像Sidecar这样的方式，在运维时与真实的应用服务集成起来。

这两种方式各有优缺点。

以软件包的方式可以和应用密切集成，有利于资源的利用和应用的性能，但是对应用有侵入，而且受应用的编程语言和技术限制。同时，当软件包升级的时候，需要重新编译并重新发布应用。

以Sidecar的方式，对应用服务没有侵入性，并且不用受到应用服务的语言和技术的限制，而且可以做到控制和逻辑的分开升级和部署。但是，这样一来，增加了每个应用服务的依赖性，也增加了应用的延迟，并且也会大大增加管理、托管、部署的复杂度。

注意，对于一些“老的系统”，因为代码太老，改造不过来，我们又没有能力重写。比如一些银行里很老的用C语言或是COBAL语言写的子系统，我们想把它们变成分布式系统，需要对其进行协议的改造以及进行相应的监控和管理。这个时候，Sidecar的方式就很有价值了。因为没有侵入性，所以可以很快地低风险地改造。

Sidecar服务在逻辑上和应用服务部署在一个结点中，其和应用服务有相同的生命周期。对比于应用程序的每个实例，都会有一个Sidecar的实例。Sidecar可以很快也很方便地为应用服务进行扩展，而不需要应用服务的改造。比如：

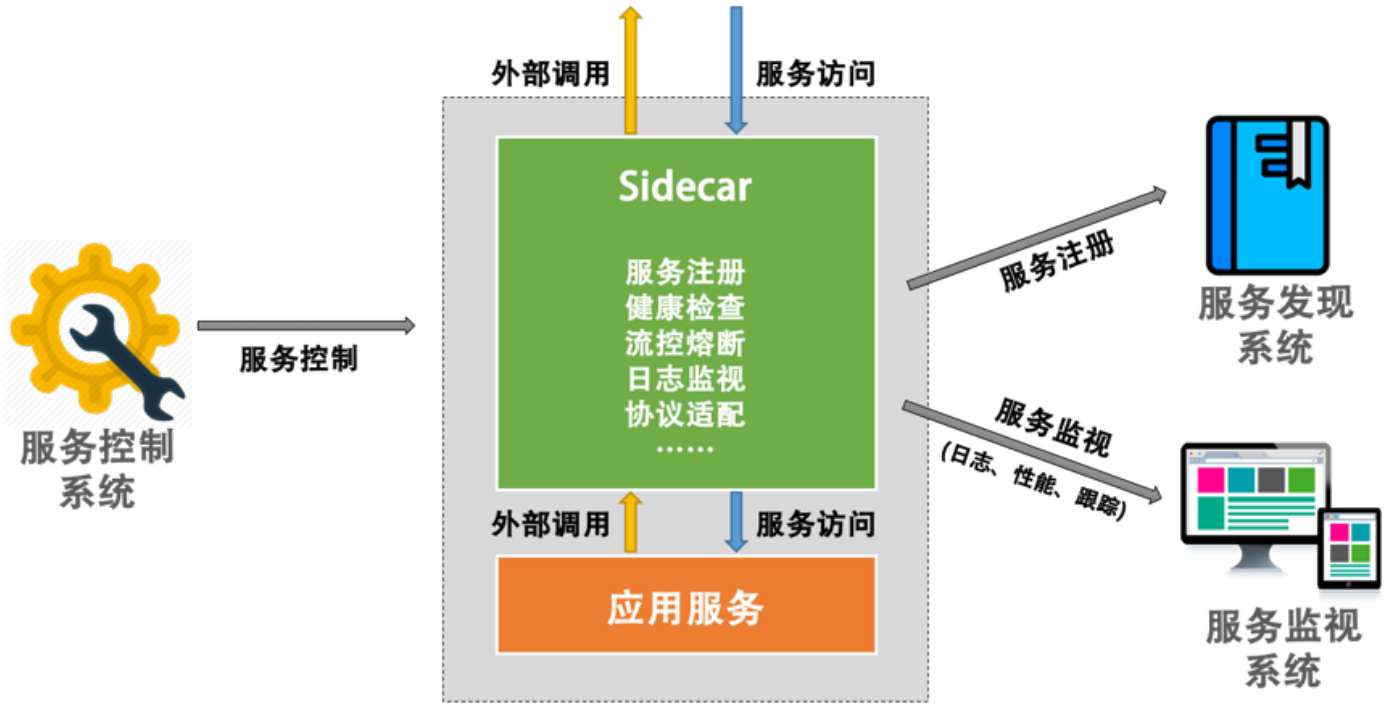
Sidecar可以帮助服务注册到相应的服务发现系统，并对服务做相关的健康检查。如果服务不健康，我们可以从服务发现系统中把服务实例移除掉。

当应用服务要调用外部服务时， Sidecar可以帮助从服务发现中找到相应外部服务的地址，然后做服务路由。

Sidecar接管了进出的流量，我们就可以做相应的日志监视、调用链跟踪、流控熔断……这些都可以放在Sidecar里实现。

然后，服务控制系统可以通过控制Sidecar来控制应用服务，如流控、下线等。

于是，我们的应用服务则可以完全做到专注于业务逻辑。



注意，如果把Sidecar这个实例和应用服务部署在同一台机器中，那么，其实Sidecar的进程在理论上来说是可以访问应用服务的进程能访问的资源的。比如，Sidecar是可以监控到应用服务的进程信息的。

另外，因为两个进程部署在同一台机器上，所以两者之间的通信不存在明显的延迟。也就是说，服务的响应延迟虽然会因为跨进程调用而增加，但这个增加完全是可以接受的。

另外，我们可以看到这样的部署方式，最好是与Docker容器的方式一起使用的。为什么Docker一定会是分布式系统或是云计算的关键技术，相信你从我的这一系列文章中已经看到其简化架构的部署和管理的重要作用。否则，这么多的分布式架构模式实施起来会有很多麻烦。

边车设计的重点

首先，我们要知道边车模式重点解决什么样的问题。

1. 控制和逻辑的分离。
2. 服务调用中上下文的问题。

我们知道，熔断、路由、服务发现、计量、流控、监视、重试、幂等、鉴权等控制面上的功能，以及其相关的配置更新，本质来上来说，和服务的关系并不大。但是传统的工程做法是在开发层面完成这些功能，这就会导致各种维护上的问题，而且还会受到特定语言和编程框架的约束和限制。

而随着系统架构的复杂化和扩张，我们需要更统一地管理和控制这些控制面上的功能，所以传统的在开发层面上完成控制面的管理会变得非常难以管理和维护。这使得我们需要通过Sidecar模式来架构我们的系统。

边车模式从概念上理解起来比较简单，但是在工程实现上来说，需要注意以下几点。

进程间通讯机制是这个设计模式的重点，千万不要使用任何对应用服务有侵入的方式，比如，通过信号的方式，或是通过共享内存的方式。最好的方式就是网络远程调用的方式（因为都在127.0.0.1上通讯，所以开销并不明显）。

服务协议方面，也请使用标准统一的方式。这里有两层协议，一个是Sidecar到service的内部协议，另一个是Sidecar到远端Sidecar或service的外部协议。对于内部协议，需要尽量靠近和兼容本地service的协议；对于外部协议，需要尽量使用更为开放更为标准的协议。但无论是哪种，都不应该使用与语言相关的协议。

使用这样的模式，需要在服务的整体打包、构建、部署、管控、运维上设计好。使用Docker容器方面的技术可以帮助你全面降低复杂度。

Sidecar中所实现的功能应该是控制面上的东西，而不是业务逻辑上的东西，所以请尽量不要把业务逻辑设计到Sidecar中。

小心在Sidecar中包含通用功能可能带来的影响。例如，重试操作，这可能不安全，除非所有操作都是幂等的。

另外，我们还要考虑允许应用服务和Sidecar的上下文传递的机制。 例如，包含HTTP请求标头以选择退出重试，或指定最大重试次数等等这样的信息交互。或是Sidecar告诉应用服务限流发生，或是远程服务不可用等信息，这样可以让应用服务和Sidecar配合得更好。

当然，我们要清楚Sidecar适用于什么样的场景，下面罗列几个。

一个比较明显的场景是对老应用系统的改造和扩展。

另一个是对由多种语言混合出来的分布式服务系统进行管理和扩展。

其中的应用服务由不同的供应商提供。

把控制和逻辑分离，标准化控制面上的动作和技术，从而提高系统整体的稳定性和可用性。也有利于分工——并不是所有的程序员都可以做好控制面上的开发的。

同时，我们还要清楚Sidecar不适用于什么样的场景，下面罗列几个。

架构并不复杂的时候，不需要使用这个模式，直接使用API Gateway或者Nginx和HAProxy等即可。

服务间的协议不标准且无法转换。

不需要分布式的架构。

小结

好了，我们来总结一下今天分享的主要内容。首先，我介绍了什么是边车模式。为了把诸如监视、日志、限流等控制逻辑与业务逻辑分离解耦，我们可以采用边车模式。与之对应的另一种实现控制逻辑的方式是库或框架。虽然相对来说边车模式资源消耗较大，但控制逻辑不会侵入业务逻辑，还能适应遗留老系统的低风险改造。

边车作为另一个进程，和服务进程部署在同一个结点中，通过一个标准的网络协议，如HTTP来进行通信。这样可以做到低延迟和标准化。同时，用Docker来打包边车和服务两者，可以非常方便部署。最后，我指出了边车模式适用和不适用的场景。下篇文章中，我们讲述服务网格。希望对你有帮助。

也欢迎你分享一下你实现服务的同时有没有实现边车模式？有没有用到Docker来打包边车和服务两者？

文末给出了《分布式系统设计模式》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

数据库扩展

秒杀Flash Sales

边缘计算Edge Computing

精选留言 17

约书亚
1524734810

出门左转，service mesh就是靠sidecar支撑起来的，随着微服务的升级换代sidecar会越来越多的提起
netflix有prana，主要是针对自己的微服务全家桶的，好像spring cloud也有对应实现

AYOU
1524703811

求一个边车模式的开源项目。。

c3po
1524879908

linkerd 边车部署

秋天
1524745589

有没有可以参考的边车模式实现？？？？

曾经的十字镐
1524721237

第一遍没有好好看，感觉和api gateway差不多，第二遍看的时候发现其中的猫腻，感谢老鼠哥

edisonhuang
1563324664

边车模式是为了解决服务控制和逻辑的分离，同时解决服务上下文的问题。和边车模式类似还有使用SDK或者框架的方式，后者相较边车模式的缺点在于需要侵入到软件代码中，当需要改造的软件较老时，使用边车模式可以不受编程语言的限制。
使用边车模式和应用软件是同机器一起部署的，虽然有跨进程调用的开销，但基本可以忽略

ZhYB
1561513914

回头一想，发现这个模式有一个问题，就是粒度比较粗，它只能做到接口级别。这样细粒度（代码片段级别）的监控、日志做不了。但是其实也无所谓，更重要更大头的熔断降级限流重试、注册、协议适配等可以做。

ZhYB
1561513518

这个有意思啊，对于复杂的分布式服务来说确实好处非常多，我竟然第一次听说。这种模式目前有公司在用了吗？控制和逻辑分离，思想很不错，实现也简单，感觉有搞头。

Sam_bo
1530660993

目前我所知道的边车模式 thanos https://github.com/improbable-eng/thanos 应该算是边车模式的一种，不确定说的是不是对的

恒修
1542505036

会有一点性能开销

Panda
1539824026

Sidecare 和Server Mesh 密切相关 例如开源项目istio

Han
1530668340

耗子叔，能否推荐个工具或框架能比较好的实现边车模式呢？

neohope
1529825534

在近期的项目中用过，新项目是基于spring cloud的，我们有些旧服务是C#写的webservice，重新封装为rest，并用了side car模式提供服务信息和监控信息，虽然与spring boot实现的服务还是有些差距，但效果还可以：）

风的叹息
1528103412

如果再有一节实践讲解就更好啦，知道原理和实践还有些路程。

free
1525307312

没个写笔记功能啊。sidecar还没用过

LIngo1990
1524738472

hadoop生态集群管理工具，cloudera manager 了解一下？

枫晴 andy

1524719318

一次调用副作用和调用多次副作用一样，这里的副作用是指什么？