

05 | 后端BaaS化（上）：NoOps的微服务

2020-04-27 蒲松洋

Serverless入门课

[进入课程 >](#)



讲述：蒲松洋

时长 18:37 大小 17.06M



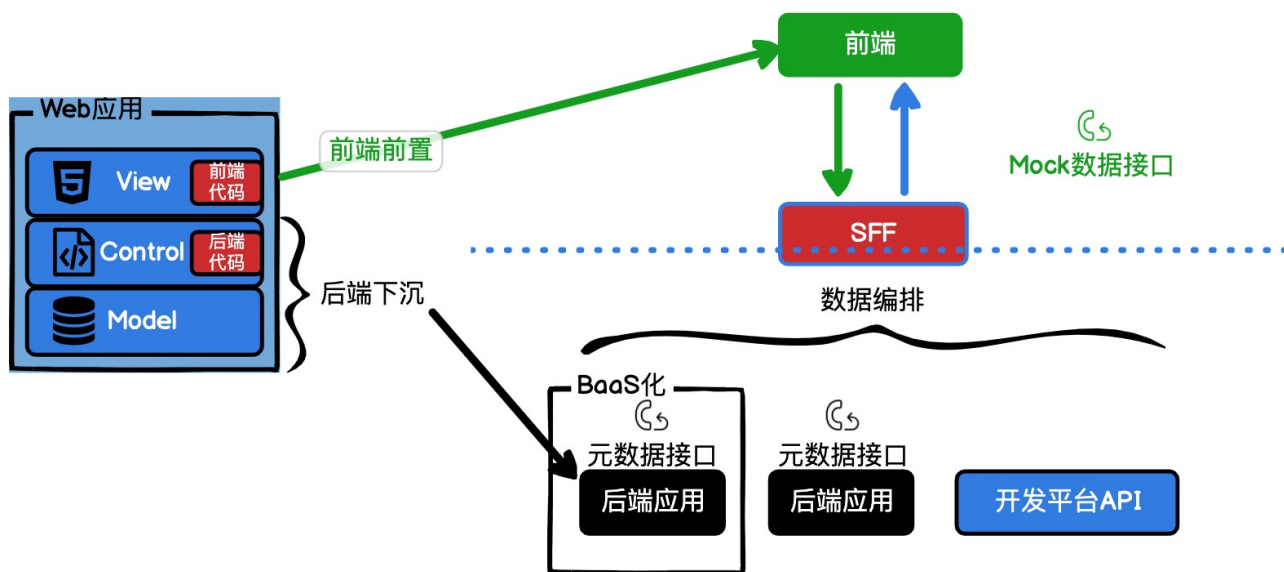
你好，我是秦粤。现在我们知道了在网络拓扑图中，只有 Stateless 节点才能自由扩缩容，而 Stateful 节点因为保存了重要数据，我们要谨慎对待，因此很难做扩缩容。

FaaS 连接并访问传统数据库会增加额外的开销，我们可以采用数据编排的思想，将数据库操作转为 RESTful API。顺着这个思路，我引出了后端应用的 BaaS 化，一句话总结，后端应用 BaaS 化就是将后端应用转换成 NoOps 的数据接口。那怎么理解这句话呢？后端应用 BaaS 化，究竟应该怎么做？接下来的几节课，我们会展开来讲。



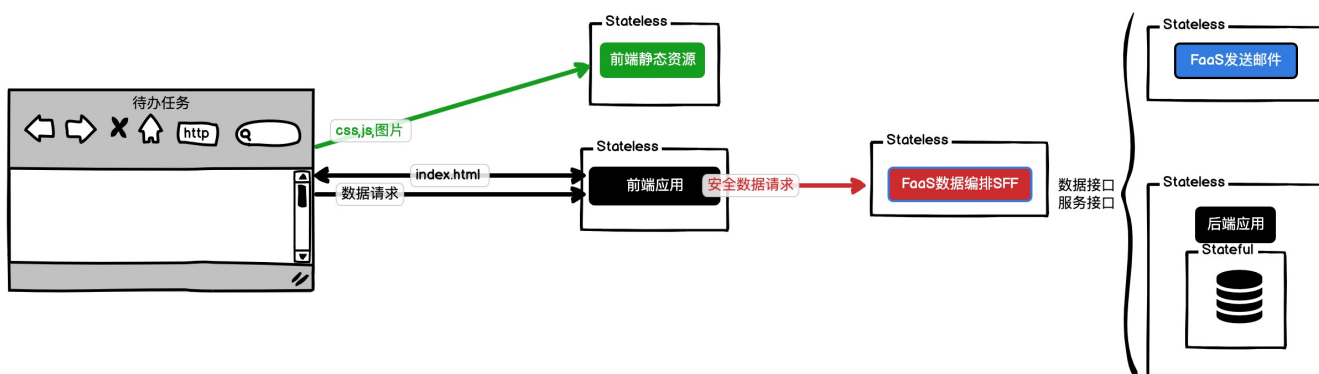
我们先回忆一下上节课的“待办任务”Web 应用，这个项目前端是单页应用，中间用了 FaaS 做 SFF 数据网关，后端数据接口还要 BaaS 化。这个案例会贯穿我们的课程，所以你

一定要动手 run 一下。为了让你对我们的项目有个宏观上的认识，我还是先交付你一张大图。



“待办任务” Web应用架构图

这个架构的优势是什么呢？我们将这个图变个形，你就更容易理解了。



架构变形示意图

咱从左往右看上面这张图。用户从浏览器打开我们网站时，前端应用响应返回 index.html；然后浏览器去 CDN 下载我们的静态资源，完成页面静态资源的加载；与此同时，浏览器也向前端应用发起数据请求；前端应用经过安全签名后，再将数据请求发送给 SFF；SFF 再根据数据请求，调用后端接口或服务，最终将结果编排后返回。

从图里你可以看到，除了数据库是 Stateful 的，其它节点都已经变成了 Stateless。如果你公司业务量不大的话，这个架构其实已经足够了。就像传统的 MVC 架构一样，单点数据

库，承载基本的并发量不是问题，而且数据也可以通过主从结构和客户端读写分离的方式来优化。

但 MVC 架构最大的问题就是累积，当一个 MVC 架构的应用，在经历长期迭代和运营后，数据库一定会变得臃肿，极大降低数据库的读写性能。而且在高并发达到一定量级，Stateful 的数据库还是会成为瓶颈。那我们可以将自己的数据库也变成 BaaS 吗？

要解决数据库的问题，也可以选择我上节课和你说的云服务商提供的 BaaS 服务，比如 DynamoDB。但云服务商 BaaS 服务究竟是怎么做到的？如果 BaaS 服务能力不全，不够满足我们的需要时怎么办？今天我就先带你看看传统的 MVC 应用中的数据库怎么改造成 BaaS。

当然，BaaS 化的过程有些复杂，这也正是我们后面需要用几节课才能跟你解释清楚的核心知识点。正如我们本节课的标题：后端应用 BaaS 化，就是 NoOps 的微服务。在我看来后端应用 BaaS 化，跟微服务高度重合，微服务几乎涵盖了我们的 BaaS 化要做的所有内容。

所以我们先来学习一下微服务是什么。

微服务的概念

微服务的概念对很多做后端同学来说并不陌生，尤其是做 Java 的同学，因为早些年 Java 就提出 SOA 面向服务架构。微服务算是 SOA 的一个子集，2014 年由 ThoughtWorks 的 Martin Fowler 提出。微服务设计之初是为了拆解巨石应用，巨石应用就是指那些生命周期较长的，累计了大量业务高度耦合和冗余代码的企业应用。

跟 Serverless 的概念还在发展中不同，微服务的概念在这么多年的发展中已经有了明确的定义了。下面是 AWS 官方的解释：

微服务是一种开发软件的架构和组织方法，其中软件由通过明确定义的 API 进行通信的小型独立服务组成，这些服务由各个小型独立团队负责。微服务架构使应用程序更易于扩展和更快地开发，从而加速创新并缩短新功能的上市时间。

那在我看来，微服务就是先拆后合，它将一个复杂的大型应用拆解成职责单一的小功能模块，各模块之间的数据模型相互独立，模块采用 API 接口暴露自己对外提供服务，然后再

通过这些接口组合出原先的大型应用。拆解的好处是，小模块便于维护，可以快速迭代，跨应用复用。

我们的 Serverless 专栏并不打算给你详细地讲解微服务，但是希望你能一定程度上了解微服务。FaaS 和微服务架构的诞生几乎是在同一时期，它俩的很多理念都是来自 12 要素 (Twelve-Factor App) [1]，所以微服务概念和 FaaS 概念高度相似，也有不少公司用 FaaS 实现微服务架构；不同的是，微服务的领军公司 ThoughtWorks 和 Netflix 到处宣扬他们的微服务架构带来的好处，而且他们提出了一整套方法论，包括微服务架构如何设计，如何拆解微服务，尤其是数据库如何设计等等。

我们后端应用 BaaS 化，首先要将复杂的业务逻辑拆开，拆成职责单一的微服务。**因为职责单一，所以服务端运维的成本会更低。**而且拆分就像治理洪水时的分流一样，它能减轻每个微服务上承受的压力。

我在 Serverless 的专栏里向你介绍微服务，主要就是想引入微服务拆解业务的分流思想和微服务对数据库的解耦方式。

微服务 10 要素

谈起微服务，很多人都要说 12 要素[2]，认为微服务也应该满足 12 要素。12 要素当初是为了 SaaS 设计的，用来提升 Web 应用的适用性和可移植性。我在做微服务初期也学习过 12 要素，但我发现这个只能提供理论认识。

所以在 2018 年 GIAC 的大会上，我将我们团队在微服务架构落地中的经验总结为**微服务的 10 要素：API、服务调用、服务发现；日志、链路追踪；容灾性、监控、扩缩容；发布管道；鉴权。**

我们回想一下  [第 1 课] 中小服的工作职责：

1. 无需用户关心服务端的事情（容错、容灾、安全验证、自动扩缩容、日志调试等等）。
2. 按使用量（调用次数、时长等）付费，低费用和高性能并行，大多数场景下节省开支。
3. 快速迭代 & 试错能力（多版本控制，灰度，CI&CD 等等）。

你有没有发现跟微服务的要素有大量的重合。API 就是 RESTful 的 HTTP 数据接口；服务调用你可以理解为就是 HTTP 请求；服务发现你可以理解为我们只能用域名调用我们的 HTTP 请求，不能用 IP；日志、容灾、监控都不难理解；链路追踪，是微服务重要的一环，因为相对传统 MVC 架构，我们一个请求在后端的调用链增长了，为了快速定位问题，我们都需要打印整个调用链路的异常栈；发布管道和鉴权，和我们的 FaaS 也有很重要的关联，我将放在下一课讲解。

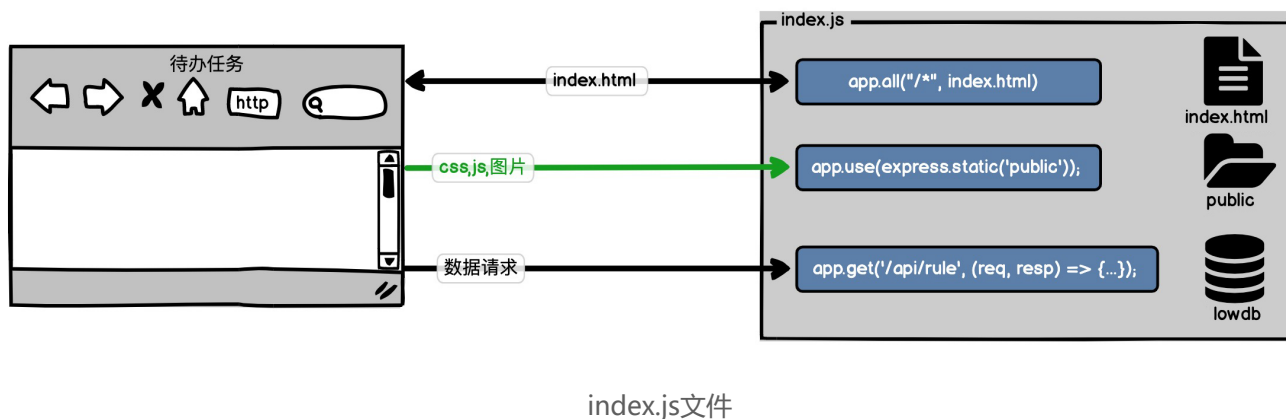


木桶效应示意图

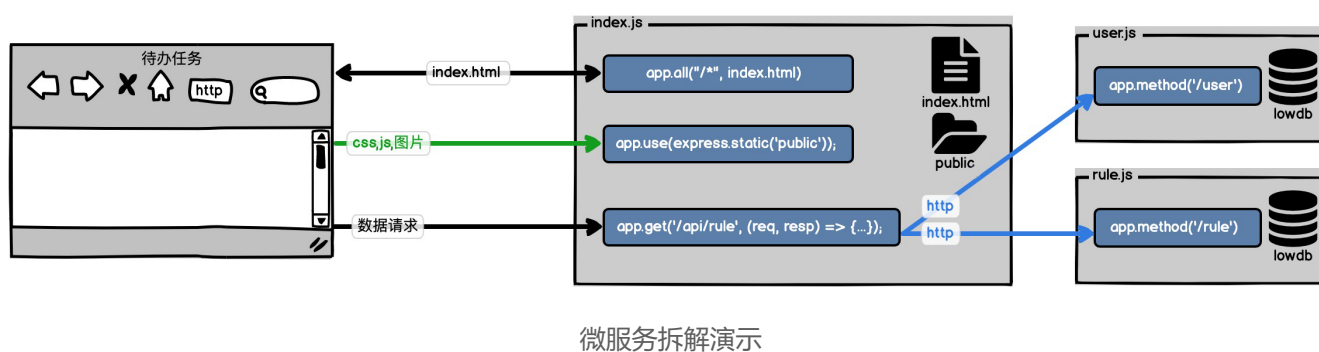
接下来，我不准备按照微服务的 10 要素——操作，但我希望可以通过 Serverless 帮你建立知识体系的索引，所以关联的技术栈我都尽量点出来，你可以自己进一步了解学习。

我们再次拿出我们的创业项目“待办任务”Web 网站，这次我将带你一起看看微服务如何让数据库解耦。

我们先分析一下“待办任务”Web 服务。上一讲末尾我的作业其实已经为你准备好了，我们一起看看 index.js 文件，这是一个典型的 MVC 架构的应用。View 层就是 index.html 和静态资源文件；Model 层，我们引入 lowdb 用一个 db.json 文件代替数据库；Control 层，就是 app.METHOD，处理 /api/* 的数据逻辑。微服务是解决后端应用的，所以我们只需要关注 Control 层。



API 的数据处理，主要有两个，一个是 /api/rule 处理待办任务的，一个是 /api/user 负责用户登录态，而且我们“待办任务”主要的数据模型也就是待办任务和用户这两个。那我们可以将后端拆分出两个微服务：待办任务微服务和用户微服务。这里我要强调下，我只是为了向你演示微服务才这样做，在实际业务中，这么简单的功能，没必要过早地拆分。



初步拆解，我们将 index.js 中的数据库移走了，而且拆分后各微服务的数据库比原来混杂在一起简单且容易维护多了。user.js 负责用户相关业务逻辑，只维护用户信息的数据库，并且暴露 RESTful 的 HTTP 方法；rule.js 负责待办任务的增删改查，只维护待办任务的数据库，并且暴露 RESTful 的 HTTP 方法；index.js 只需要负责将请求 HTTP 返回给数据编排。

HTTP 协议要满足我们的服务调用与发现，发布的 user.js 和 rule.js 必须使用域名，例如 api.jike-serverless.online/user 和 api.jike-serverless.online/rule。这样新扩容的机器 IP，只需要注册到这个域名下就可以被服务发现 IP 并调用了。

我们现在拆分后只是将数据库从 index.js 移出，分给到了 user.js 和 rule.js，但两个节点到目前为止数据库仍然是 Stateful 的。我们如何再让这两个数据库节点变成 Stateless 呢？你可以按一下暂停键思考一下。

我们先想想，对于 rule.js 这个微服务来说，我要再扩容一个新的实例，而且让两边的实例保持一致，那么我们是否只需要让新的数据库和旧的数据库保持同步更新就可以了呢？

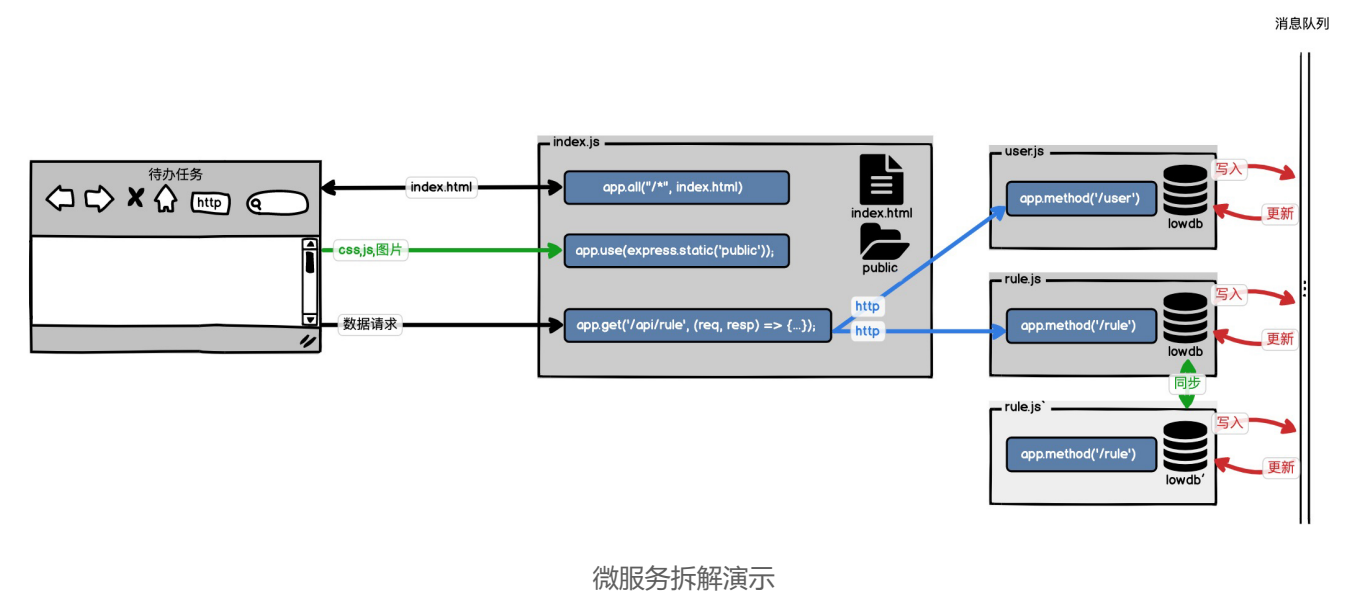
解耦数据库

没错，其实要解决这个问题的核心就是让数据启动时，可以更新到最新的数据库。在其中一个数据库更新时，通知另外一个数据库更新。

这里就要引出一个关键的 Stateful 对象：消息队列[3]。它是一个稳定的绝对值得信赖的 Stateful 节点，而且对你的应用来说消息队列是全局唯一的。解耦数据库的思路就是，通过消息队列解决数据库和副本之间的同步问题，有了消息队列，剩下的就简单了。

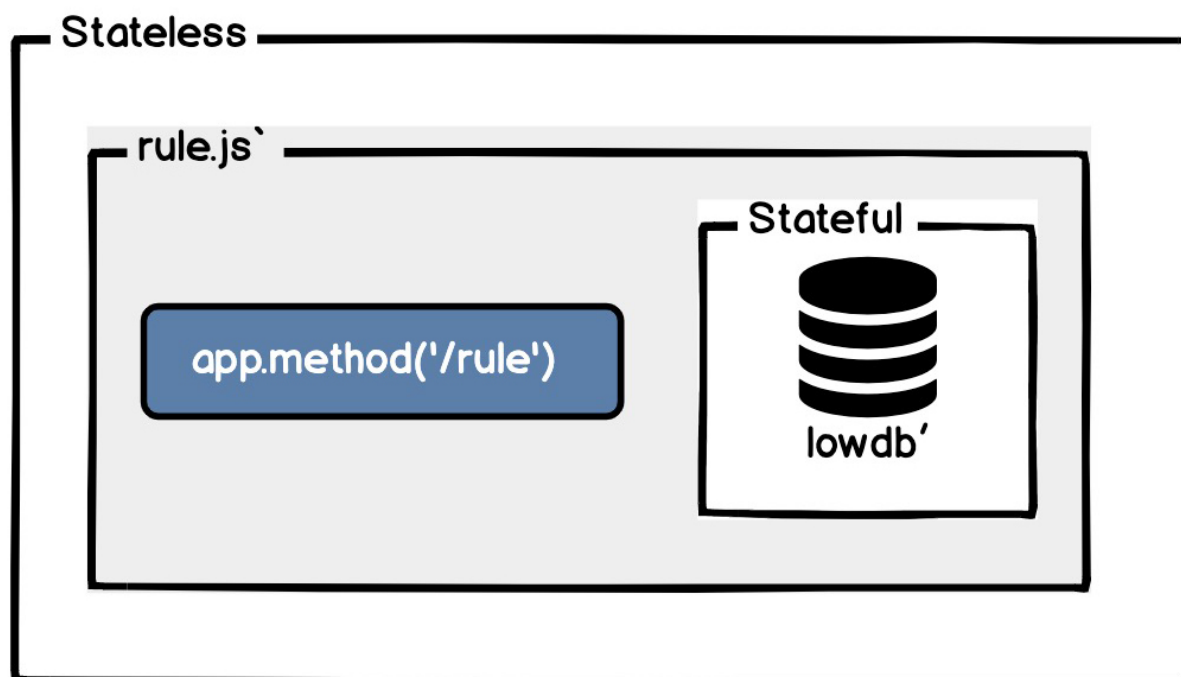
我们每个微服务中的数据库，例如 MySQL，写的操作都会产生 binary log，通过额外的进程将 binary log 变更同步到消息队列，并监听消息队列将 binary log 更新在本地执行，修改 MySQL。比较著名的解决方案就是领英开源的 Kafka，现在云服务商基本也都会提供 Kafka 服务。当然数据库和副本之间会有短暂的同步延迟问题，但问题其实也不大，因为我们通常对数据库进行写操作时也会锁表。

如果你要细究这个问题，那就会碰到分布式架构无法同时满足的 CAP[4] 课题。我们在此也不深入展开 CAP 话题了，目前异步同步的时间差在很多场景下我觉得是可以接受的。



拿我们自己的例子来说也很简单，我们用消息队列 RocketMQ，写操作我们都写入 RocketMQ，rule.js 进程中我们监听 RocketMQ，一旦有 rule 写入的消息，我们就更新我们的 lowdb 数据。围绕消息队列建立的同步机制，让每个微服务的数据库和它的扩容副本

之间自动同步了。对于微服务来说，它本身的数据库还是 Stateful 的，但在微服务外部看来，这个微服务是 Stateless 的。如下图所示：



同步机制

跟传统的主从数据库方式不同的是，我们每个微服务的单点实例都是独享一个数据库的，rule 这个微服务单点可以对外提供所有待办任务的 RESTful API 接口。

你要知道消息队列，例如 RocketMQ 的可靠性是 99.99999999%，而我们常用的虚拟机 ECS 的可靠性也只不过 99.95%。在高并发架构的设计中，我们通常会要求 Stateful 节点一定要稳定，而且越少越好，所以消息队列，对于微服务和 FaaS 来说，都是可以作为支撑业务的核心。我们依赖消息队列，会让整个架构更稳定。

总结

我们为了避免在 FaaS 中直接操作数据库，而将数据库操作变成 BaaS 服务。为了理解 BaaS 化具体的工作，我们引入了微服务的概念。微服务先将业务拆解成单一职责和功能的小模块便于独立运维，再将小模块组装成复杂的大型应用程序。这一点上跟我们要做的后端应用 BaaS 化相似度非常高。所以参考微服务，我们先将 MVC 架构的后端解成了两个微服务，再用微服务解耦数据库的操作，先将数据库拆解成职责单一的小数据库，再用消息队列同步数据库和副本，从而将数据变成了 Stateless。

现在我们来回顾一下这节课的重要知识点。

1. 微服务的概念：它就是将一个复杂的大型应用拆解成职责单一的小功能模块，各模块之间数据模型相互独立，模块采用 API 接口暴露自己对外提供的服务，然后再通过这些 API 接口组合成大型应用。这个小功能模块，就是微服务。
2. 微服务 10 要素：API、服务调用、服务发现；日志、链路追踪；容灾性、监控、扩缩容；发布管道；鉴权。这跟我们要做的 BaaS 化高度重合，我们可以借助微服务来实现我们的 BaaS 化。
3. 解耦数据库：通过额外进程让数据库与副本直接通过消息队列进行同步，所以对于微服务应用来说，只需要关注自身独享的数据库就可以了。微服务通过数据库解耦，将后端应用变成 Stateless 的了，但对后端应用本身而言，数据库还是 Stateful 的。

作业

由于 Kafka 云服务比较贵，所以这节课的作业我们采用表格存储[5]来模拟 Kafka，以实现 DB 的同步功能。**你可以尝试自己部署一下 rule-faas 到一个新的域名。**下面是具体的代码地址：

仓库地址：🔗 <https://github.com/pusongyang/todolist-backend/tree/lesson05>

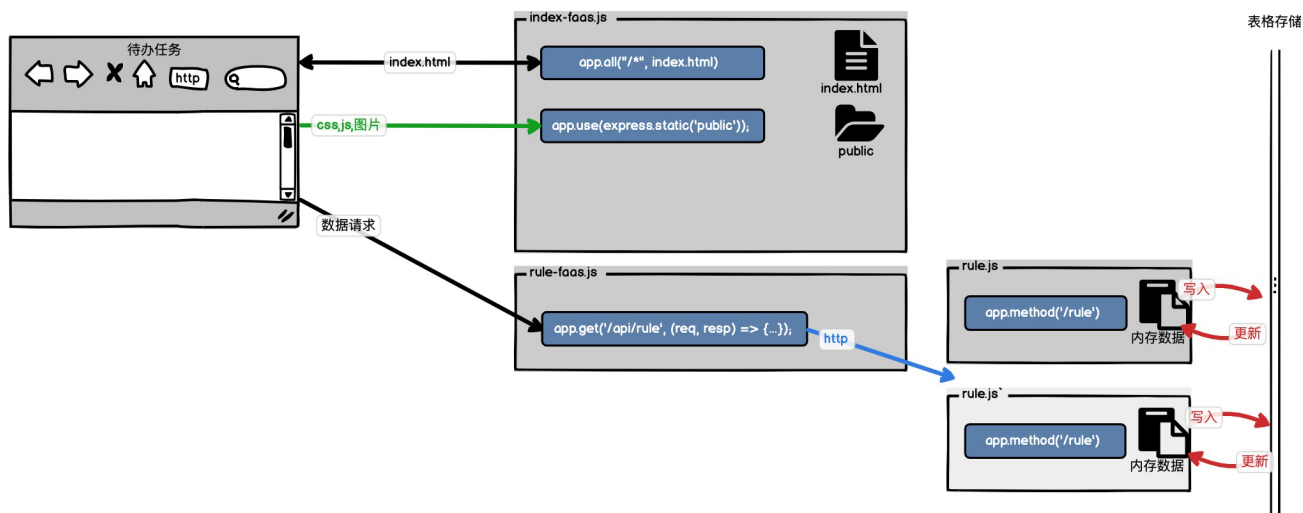
前端地址：🔗 <https://github.com/pusongyang/todolist-frontend/tree/lesson05>

实现后效果：🔗 <http://lesson5.jike-serverless.online/list>

你可以根据自己的域名，替换一下前端文件中的这个地址：🔗 <http://faas.jike-serverless.online/api/rule>，或者用我的前端代码🔗 [lesson5](#)分支修改

/src/pages/ListTableList/service.ts 中的地址，自己 npm run build 一下，替换掉项目 public 目录，压缩上传到函数服务目录。

这次的部署会有些复杂，我们要部署两个 FaaS 服务，一个是 rule-faas.js，用来处理 /api/rule；另一个是 index-faas.js，用来处理我们的静态资源请求。原理图如下所示：



部署示意图

另外.aliyunConfig 文件格式如下，你需要填入自己的配置情况，切记这个文件不可以上传到 Git 仓库：

复制代码

```
1 // .aliyunConfig 文件，保存密钥，切记不可以上传Git
2 const endpoint = "https://rule.cn-shanghai.ots.aliyuncs.com";
3 // AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
4 const accessKeyId = "AccessKey";
5 // SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
6 const accessKeySecret = "SecretKey";
7 // 在数据链表中查看
8 const instancename = "rule";
9 const tableName = 'todos';
10 const primaryKey = [{ 'key': 'list' } ];
11
12
13 module.exports = {endpoint, accessKeyId, accessKeySecret, instancename, tableName};
14
```

期待你的作业和思考，如果今天的内容让你有所收获，也欢迎你把文章分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://en.wikipedia.org/wiki/Twelve-Factor_App_methodology

[2] https://12factor.net/zh_cn/

[3] https://help.aliyun.com/document_detail/29532.html?spm=5176.11065259.1996646101.searchclickresult.6936139bcS8BIU

[4] <https://www.ruanyifeng.com/blog/2018/07/cap.html>

[5] <https://ots.console.aliyun.com/>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 原理：FaaS应用如何才能快速扩缩容？

下一篇 06 | 后端BaaS化（中）：业务逻辑的拆与合

精选留言 (5)

写留言



我来也

2020-04-27

哈哈，原来老师也觉得云厂商提供的kafka云服务贵。
我对这个深有体会！

我看领导买的一个阿里云kafka云服务，要4K+/月，配置也就一般吧，可能监控做的比较好吧。...

展开

作者回复: 用CaaS服务，自己搭建一套Kafka还便宜一些。现在程序员可以调度的机器资源还是很充沛的。

2

3



qinsi

2020-04-28

请问数据库解耦时副本数据库允许数据写入吗？如果允许写入的话会不会因为写入的数据冲突导致数据不一致呢？

展开

作者回复: 数据副本只是为了区分的叫法，当然允许写入了。Binary log有一定的容错性，要考虑事务逻辑。但如果是银行或者支付的场景，要解决强一致性。可以直接写消息队列，再去消费数

据。



吴科

2020-04-28

老师，关于用mq实现数据库同步的问题，如果要求数据强一直都业务场景很难实现啊，比如秒杀，通过mq同步不及时，用户会查到不同的库存，造成多卖的情况呢

作者回复: 你说的这种数据强一致性的场景，要直接用消息队列去维护一致性问题，生产/消费模型，还要用到事务完成和回滚。

其实有些超纲，架构设计要考虑自己业务发展，如果你们发展到这个程度，架构配套肯定也要跟进。我举例的业务场景下，没有那么强的一致性。



我来也

2020-04-27

感觉现在的学习门槛越来越高了。😞

0.要有自己的域名,最好还是备案过的.

1.需要会排查依赖包缺失的问题.

2.需要会开通简单的云服务....

展开

作者回复: 目前阿里云有个云开发平台，正在内测。最后一课。我会放出来给大家介绍。

门槛的确会越来越高，带大家走进到serverless的深水区，层层深入。然后，再回头看serverless，你会有不一样的视野。



电光火石

2020-04-27

老师好，在后台服务baas化的过程中，关于数据库跟以前微服务的方式有些不一样，微服务的各个节点是共享数据库的，但是baas化中，每个节点有自己的数据库，而且数据库的内容是一样的。这样是否带来2个问题：

1. 在数据量很大的情况下，每个节点都有一样的数据库，是否会使得数据存储量会翻几倍，成本会很高...

展开

作者回复: 微服务标准的做法是一个微服务实例独享一个数据库实例的。当然你业务量不大, 可以多个微服务实例共享一个数据库实例。

微服务架构在docker流行后才火, 就是因为docker可以让计算资源进一步碎片化, 所以成本并不会很高。

数据实例的启动时, 通常是先从现有数据库副本创建, 再根据消息队列同步后续数据。启动实例过程是秒级的。

微服务架构, 本身就是通过拆解将复杂的业务问题变成一个个小的职责单一的小服务。运维的复杂度会上升, 但是整体性能并不会下降多少。

你可以想想, 现在的京东, 淘宝这样的网站, 背后都是微服务架构的。如果你用单体应用开发这么庞大的一个应用, 这么多团队和人员参与, 怎么去运维和开发。

