03 | 原理：FaaS的两种进程模型及应用场景

2020-04-22 蒲松洋

Serverless入门课 进入课程 >

讲述：蒲松洋
时长 19:10 大小 17.56M

你好，我是秦粤。上一讲我们通过一个 Node.js 纯 FaaS 的 Serverless 应用，给你介绍了 Serverless 引擎盖下的运作机制，总结来说，FaaS 依赖分层调度和极速冷启动的特性，在无事件时它居然可以缩容到 0，就像我们的声控灯一样，有人的时候它可以亮起来，没人的时候，又可以自动关了。

听完了原理，我估计你肯定会问，FaaS 这么好，但是它的应用场景是什么呢？今天我们就来一起看下。不过，想要理解 FaaS 的应用场景，我们就需要先理解 FaaS 的进程模型，这也是除了冷启动之后的另外一个重要概念。

FaaS 进程模型

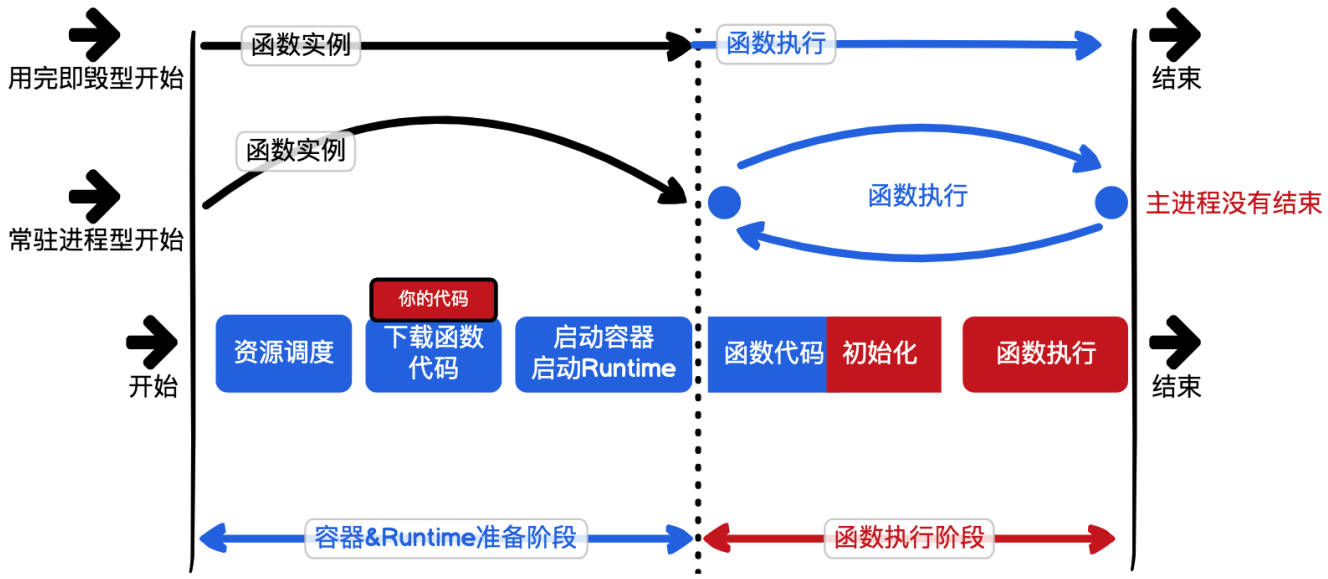
咱先回想一下上节课的 FaaS 的冷启动过程，我们知道容器和 Runtime 准备阶段都是由云服务商负责的，我们只需要关注具体的函数执行就可以了。而函数执行在 FaaS 里是由“函数服务”负责的，当函数触发器通知的“事件”到来时，函数服务就会根据情况创建函数实例，然后执行函数。当函数执行完之后，函数实例也随之结束自己的使命，FaaS 应用缩容到 0，然后开始进入节能模式。

上面这套逻辑是我们上节课讲的，课后有同学就问，函数执行完之后实例能否不结束，让它继续等待下一次函数被调用呢？这样省去了每次都要冷启动的时间，响应时间不就可以更快了吗？

是的，本身 FaaS 也考虑到了这种情况，所以从运行函数实例的进程角度来看，就有两种模型。我也画了张图，方便你理解。

用完即毁型：函数实例准备好后，执行完函数就直接结束。这是 FaaS 最纯正的用法。

常驻进程型：函数实例准备好后，执行完函数不结束，而是返回继续等待下一次函数被调用。**这里需要注意，即使 FaaS 是常驻进程型，如果一段时间没有事件触发，函数实例还是会被云服务商销毁。**



模型示意图

这两个模型其实也对应两种不同的应用场景。我举个例子，比如你要把我们第一讲中的“待办任务”应用部署上线。还记得小程同学吧，他完成了第一个版本，他用 Express.js[1] 框架开发的 MVC 架构，View 层他采用流行的 React[2]，并且使用了 Ant Design Pro[3]

React 组件库。Model 数据库采用 MongoDB。小程的第一个版本，就是一个典型的传统 Web 服务。

从可控性和改造成本角度来看 Web 服务，服务端部署方案最适合的还是托管平台 PaaS 或者自己搭服务跑在 IaaS 上。正如我上一讲所说，使用 FaaS 就必须在 FaaS 的条件限制内使用，最佳的做法应该是一开始就选用 FaaS 开发。

但是小程的运气比较好，我们查了一下文档，发现 FaaS 的 Node.js 的 Runtime 是支持 Express 的，所以我们只需少量修改，**小程的第一个版本就可以使用 FaaS 的常驻进程方案部署。**

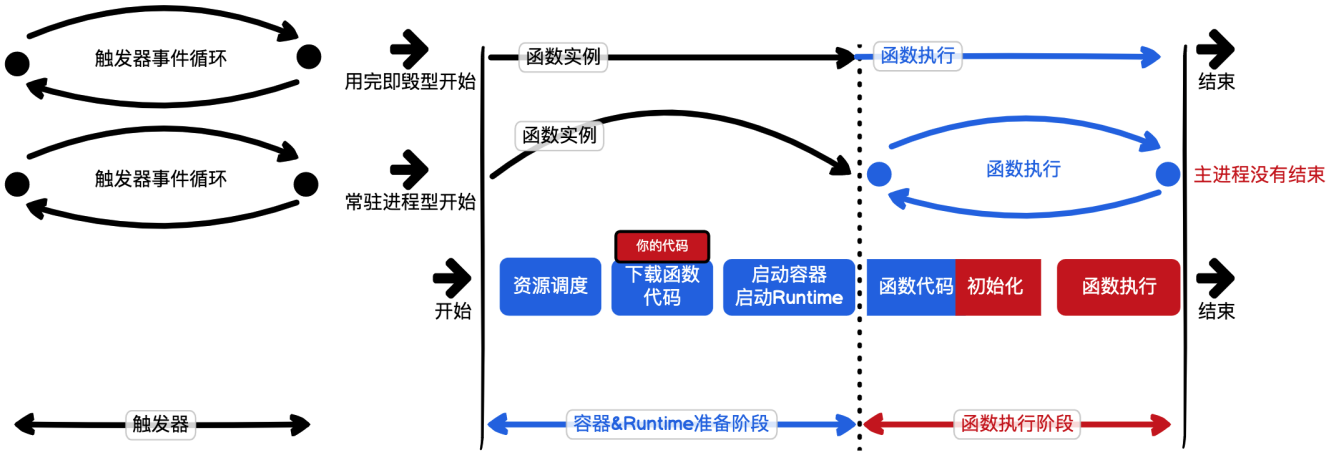
这里我要做个对比，在之前，假设没有 FaaS，我们要将应用部署到托管平台 PaaS 上。启动 Web 服务时，主进程初始化连接 MongoDB，初始化完成后，持续监听服务器的 80 端口，直到监听端口的句柄关闭或主进程接收到终止信号。当 80 端口和客户端建立完 TCP 链接，有 HTTP 请求过来，服务器就会将请求转发给 Web 服务的主进程，这时主进程会创建一个子进程来处理这个请求。

而在 FaaS 常驻进程型模式下，首先我们要改造一下代码，Node.js 的 Server 对象采用 FaaS Runtime 提供的 Server 对象，然后我们把监听端口改为监听 HTTP 事件。启动 Web 服务时，主进程初始化连接 MongoDB，初始化完成后，持续监听 HTTP 事件，直到被云服务商控制的父进程关闭回收。

当 HTTP 事件发生时，我们的 Web 服务主进程跟之前一样，创建一个子进程来处理这个请求事件。主进程就如我们上图中绘制的那个蓝色的圆点，当 HTTP 事件发生时，它创建的子进程就是蓝色弧形箭头，当子进程处理完后就会被主进程回收。

在我看来，常驻进程型就是为了传统 MVC 架构部署上 FaaS 专门设计的。数据库也可以使用原来的 DB 连接方式，不过这样做会增加冷启动的时间（我特意在图中用曲线代表时间增加），从而导致第一次请求长延迟甚至失败。比较适合的做法是我们[第 1 课] 中，讲 Serverless 架构时说的，数据持久化采用 BaaS 服务。

那么我们能否用用完即毁型来部署小程的这个 MVC 架构的 Web 服务呢？可以，但是我不推荐你这样做。因为用完即毁型对传统 MVC 改造的成本太大。



模型示意图

说到这里，我们再将上面对比两个模型的示意图镜头再拉远一点，加上 HTTP 触发器看看。其实从另外一个角度看，触发器就是一个常驻进程型模型一直在等待，只不过这个触发器是由云服务商处理罢了。

这里我再啰嗦强调下，还是我们上一讲说的，FaaS 只是做了极端抽象，云服务商通过技术手段帮助开发者屏蔽了细节，让他们尽量只关注代码本身。

所以，在用完即毁型中，我们只要将 MVC 的 Control 层部署到函数执行就可以了。这也意味着我们要将我们的 MVC 架构的 Control 函数一个个拆解出来部署，一个 HTTP 请求对应一个 Control 函数。Control 函数实例启动时连接 MongoDB，一个请求处理完后直接结束。你如果要提升 Control 函数的冷启动时间，Model 层同样要考虑 BaaS 化改造。这里你听着可能有点陌生，没关系，后面我会通过代码给你演示，你到时候再理解也不迟。

现在，理解了两种类型，我们再来看看 FaaS 是怎么收费的，以及常驻型进程这种模式是不是官方会多收费。云服务商 FaaS 函数服务的收费标准各不相同，但他们都会提供一定的免费额度。我给你归纳下 FaaS 的收费标准，主要有两个维度：调用函数次数和函数耗时。

> 调用函数次数，函数每次被事件触发，计数器加一。例如我们 Hello World 例子的 index.js 文件的 handler 函数，它每调用一次，计数就加一。这种模式因为不占资源，所以资源利用率高、收费低。
>
> 函数耗时，说的是函数执行的运行时长，它的计算单位是 CU-S，也就是 CPU 运行了多少秒。

例如我们上面“待办任务”改造的常驻进程型和用完即毁型，多数情况下其实他们两个的函数耗时是一样的。这里可能有些绕，需要给你解释一下。

常驻进程型改造后主要占用的是内存，而 FaaS 收费的是 CPU 计算时间，也就是说常驻进程的模式并不会持续收费。但常驻型应用的冷启动时间会增加，所以我们要尽量避免冷启动，避免冷启动通常又需要做一些额外的工作，比如定时触发一下实例或者购买预留实例，这地方就会增加额外的费用了。这样听起来，是不是觉得常驻进程型改造 MVC 应用用起来很别扭？是的，我们前面也说了，常驻进程模式就是为了传统 MVC 架构部署上 FaaS 专门设计的，算是一种权宜之计吧。

用完即毁型改造后，同样冷启动时间会增加。但是冷启动时间是云服务商负责的。我们 Control 函数的执行时间，和 MVC 部署在 FaaS 中 Control 的执行时间是一样的。每个请求都增加了冷启动时间，响应时间会更长一些，但我们不用考虑额外的成本。那学到这儿，相信你也可以感觉到，用完即毁型也不太适合传统 MVC 架构改造，也是一种权宜之计。

接下来，我们就继续把焦点放到用完即毁型上，来具体看看它可以用在哪些更加自然的场景里。

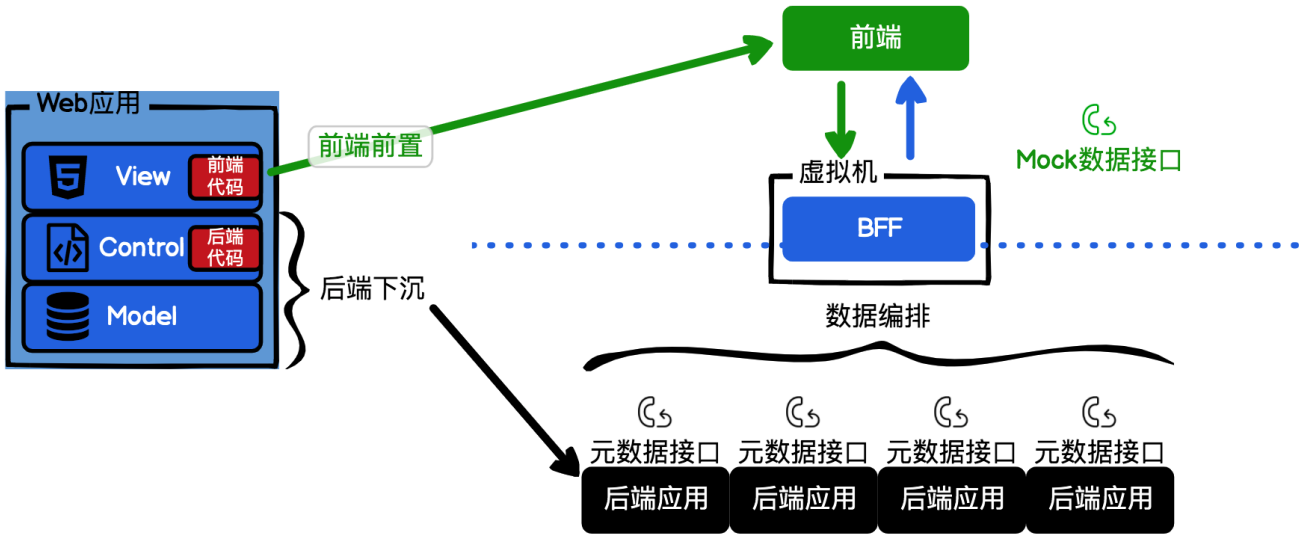
数据编排

我们做开发的多多少少都知道，目前最成功最广泛的设计模式就是 MVC 模式。但随着前端 MVVM 框架越来越火，前端 View 层逐渐前置，发展成 SPA 单页应用。后端 Control 和 Model 层逐渐下沉，发展成面向服务编程的后端应用。

这种情况下，前后端更加彻底地解耦了，前端开发可以依赖 Mock 数据接口完全脱离后端限制，而后端的同学则可以面向数据接口开发，但这也产生了高网络 I/O 的数据网关层。

Node.js 的异步非阻塞和 JavaScript 天然亲近前端工程师的特性，自然地接过数据网关层。因此也诞生了 Node.js 的 BFF 层 (Backend For Frontend)，将后端数据和后端接口编排，适配成前端需要的数据结构，提供给前端使用。

我们的程序员好朋友小程也跟进了这个潮流，将“待办任务”Web 服务重构成了第二个版本。他将原先的应用拆解成了 2 个项目：前端项目采用 React+AntDesignPro+Umi.js[4] 的单页应用，后端项目还是采用 Express。我们本专栏的示例也采用这个技术架构一步一步教你在云上部署 SPA+FaaS 混合框架演进。



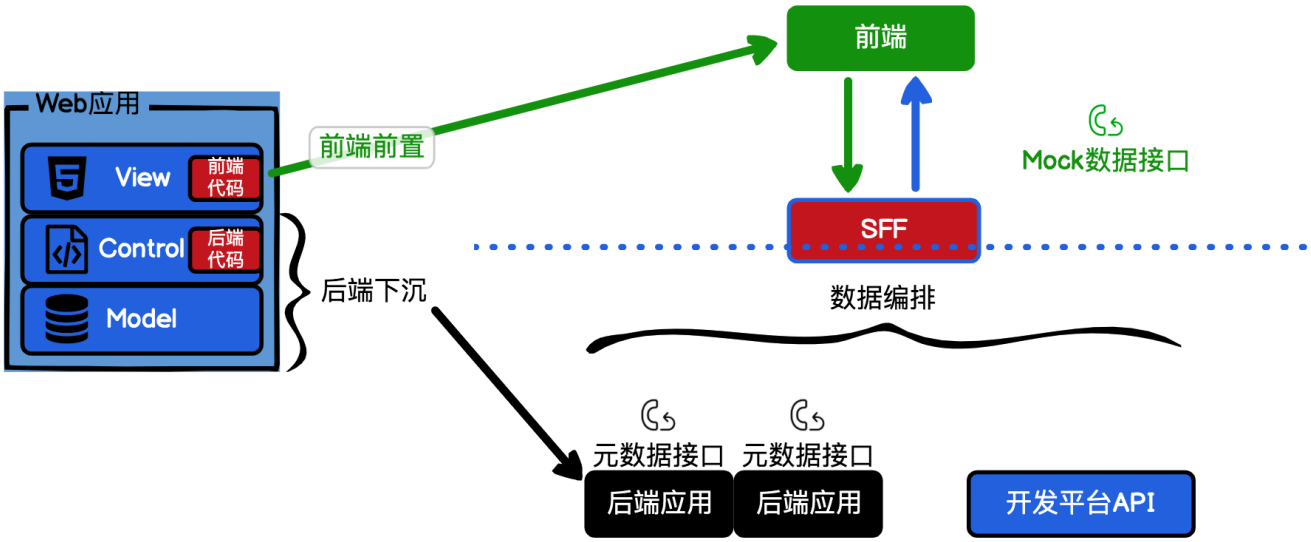
BFF示意图

如上图所示，BFF 层充当了中间胶水层的角色，粘合前后端。未经加工的数据，我们称为元数据 Raw Data，对于普通用户来说元数据几乎不可读。所以我们需要将有用的数据组合起来，并且加工数据，让数据具备价值。对于数据的组合和加工，我们称之为**数据编排**。

BFF 层通常是由善于处理高网络 I/O 的 Node.js 应用负责。传统的服务端运维 Node.js 应用还是比较重的，需要我们购买虚拟机，或者使用应用托管 PaaS 平台。

因为 BFF 层只是做无状态的数据编排，所以我们完全可以用 FaaS 用完即毁型模型替换掉 BFF 层的 Node.js 应用，也就是最近圈子里老说的那个新名词 SFF（Serverless For Frontend）。

好，到这儿，我们已经理解了 BFF 到 SFF 的演进过程，现在我们再串下新的请求链路逻辑。前端的一个数据请求过来，函数触发器触发我们的函数服务，我们的函数启动后，调用后端提供的元数据接口，并将返回的元数据加工成前端需要的数据格式。我们的 FaaS 函数完全就可以休息了。具体如下图所示。



SFF示意图

另外，除了我们自己的后端应用数据接口，互联网上还有大量的数据供我们使用，比如疫情期间，你要爬取下各个地区的疫情数据、天气数据，这些工作，也都可以放到 FaaS 上轻松搞定。并且，基本还能免费，因为目前各大云服务商都提供了免费的额度，这个我刚给你讲过了。

编排后端接口、编排互联网上的数据，这俩场景我想你也很容易想到。不过，我觉得，编排云服务商的各种服务才能让你真正体会到那种触电的感觉。我第一次体验之后，就对我同事说，“变天了，真的变天了，喊了这么多年的云计算时代真的来了”。

服务编排

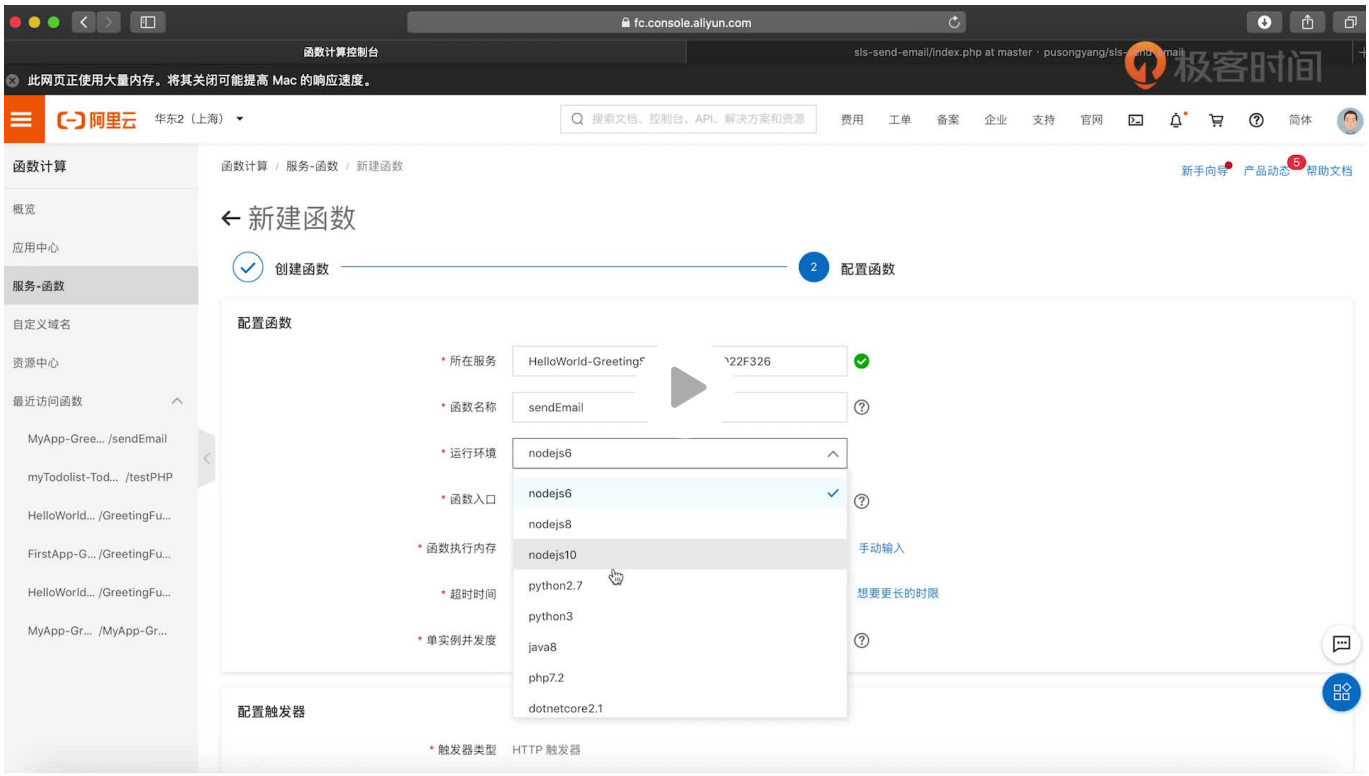
服务编排和数据编排很像，主要区别是对云服务商提供的各种服务进行组合和加工。在 FaaS 出现之前，就有服务编排的概念。但服务编排受限于服务支持的 SDK 语言版本，常见的情况是我们用 yaml 文件或命令行来编排服务。我们要使用这些服务或 API，都要通过自己熟悉的编程语言去找对应的 SDK，在自己的代码中加载 SDK，使用秘钥调用 SDK 方法进行编排。就和数据编排一样，服务端运维部署成本非常高。而且如果没有 SDK，则需要自己根据平台提供的接口或协议实现 SDK。

现在有了 FaaS，FaaS 拓展了我们可以使用 SDK 边界。这是什么意思呢？比如小程的“待办任务”Web 服务需要发送验证码邮件。我们可以用一个用完即毁型 FaaS 函数，调用云服务商的 SDK 发送邮件，再用一个常驻进程型 FaaS 函数生成随机字符串验证码，生成后

记录这个验证码，并且调用发送邮件的 FaaS 将验证码发给用户邮箱。用户验证时，我们再调用常驻进程型 FaaS 的方法校验验证码是否正确。

我还是用阿里云来举例，我们查阅阿里云的邮件服务文档，发现它只支持 Java、PHP 和 Python 的 SDK。我们一直都是在讲 Node.js，这里没有 Node.js 的 SDK，怎么办？如果我们根据阿里云邮箱服务的文档，自己开发 Node.js 的 SDK，那肯定是饶了弯路，废了没用的力气。

因为我们发送邮件的用完即毁型 FaaS 函数功能很单一，所以我们完全可以参考邮件服务的 PHP 文档，就用 PHP 的 SDK 创建一个 FaaS 服务来发送邮件的。你会发现使用 PHP 邮件服务的成本居然如此之低。



你会看到在这个例子中，我用了我并不太熟悉 PHP 语言编排了邮件发送服务。不知道你意识到没有，这个也是 FaaS 一个亮点：语言无关性。它意味着你的团队不再局限于单一的开发语言了，你们可以利用 Java、PHP、Python、Node.js 各自的语言优势，混合开发出复杂的应用。

FaaS 服务编排被云服务商特别关注正是因为它具备的这种开放性。使用 FaaS 可以创造出各种各样复杂的服务编排场景。而且还是语言无关，大大提升了云服务商各种服务的使用场景。当然，这对开发者也提出了要求，它要求开发者去更多地了解云服务商提供的各种服务。

甚至我还知道，西雅图就有创业团队利用 FaaS 服务编排能力做了一套开源框架：Pulumi[5]，并且还拿到了融资。感兴趣的话，你可以去他们的官网看看。

总结

好，到这里，我们这节课的内容就讲完了。我再来总结一下这节课的关键点。

1. FaaS 的进程模型有两种：常驻进程型和用完即毁型。常驻进程型是为了适应传统 MVC 架构设计的，它看起来并不自然。如果你从现在开始玩 FaaS 的话，我当然首选推荐用完即毁型，它可以最大限度发挥 FaaS 的优势。
2. 追溯历史，我给你梳理了前后端分离发展出的 BFF，然后 BFF 又可以被 SFF 替代。不管是内部的接口编排，还是外部一些数据的编排，FaaS 都可以发挥出极大优势，你看看我视频演示的例子就懂了。
3. 从数据编排再进一步，我们可以利用 FaaS 和云服务商云服务的能力，做到服务编排，编排出更加强大的组合服务场景，提升我们的研发效能。并且通过我这么长时间的体验，我还想感叹说，依赖云服务商的各种能力，再通过 FaaS 编排开发一个项目时，往往可以做到事半功倍。

作业

今天的作业和上一讲类似，我视频中给你做了个简单的 Demo。你可以随便找个云平台去 run 一下试试，百闻不如一见，体验完之后，你可以在留言区谈谈你的感想。另外，如果今天这节课让你有所收获，也欢迎你把它分享给更多的朋友。

我 Demo 中的代码地址：https://github.com/pusongyang/sls-send-email

参考资料

[1] Express 是 Node.js 著名的 Web 服务框架 <https://expressjs.com/>。

[2] React 是 Facebook 开源的 MVVM 框架 <https://zh-hans.reactjs.org/>。

[3] AntDesignPro 是蚂蚁开源的 React 组件库 <https://pro.ant.design/>。

[4] Umi.js 是蚂蚁开源的 React 企业级解决方案脚手架 <https://umijs.org/>。

[5] Pulumi <https://pulumi.io/>

上一篇 02 | 原理：通过一个案例，理解FaaS的运行逻辑

精选留言 (4)

写留言

我来也
2020-04-22

最近几天实战了下阿里云的函数计算服务.
使用golang按着官方文档,实现了几个常驻进程模型的服务.
也照着老师的操作,建了node.js和python的函数.

相比之下,python和node.js的冷启动时长确实比较短,我这边看冷启动时接口的响应耗时...
展开

作者回复: 必须给你手动点赞，写的非常详细。云服务商承诺的回收时间，目前其实是不确定的。能够承诺时间的只有AWS（所以我的文章中没有给出具体回收时间）。通常阿里云1分钟到10分钟，没有请求就会被回收，不过1分钟是可以承诺的。因为冷启动的时间很快，所以通常影响不大。如果对冷启动增加的几百毫秒比较敏感的场景，还是建议使用CaaS服务。

3

一步
2020-04-22

常驻型应用的冷启动时间会增加, 这里为什么会增加呢？ 不应该是减少吗？ 只需要第一次启动后，长驻内存不就行了吗?
展开

作者回复: 常驻进程启动后，就没有冷启动过程了。冷启动指的是函数实例的启动过程。

6 2

一步
2020-04-22

阿里云 有个 serverless 工作流 是不是就是 一个阿里云提供的云服务的编排工具?

作者回复: 手动给你点个赞，悟性很高。我后面也会介绍到，除了代码编排外，还可用事件流编排。

1

Marooned。

2020-04-22

因为用完即毁型对传统 MVC 改造的成本太大 为什么会大呢?

1