03-原理：FaaS的两种进程模型及应用场景

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。

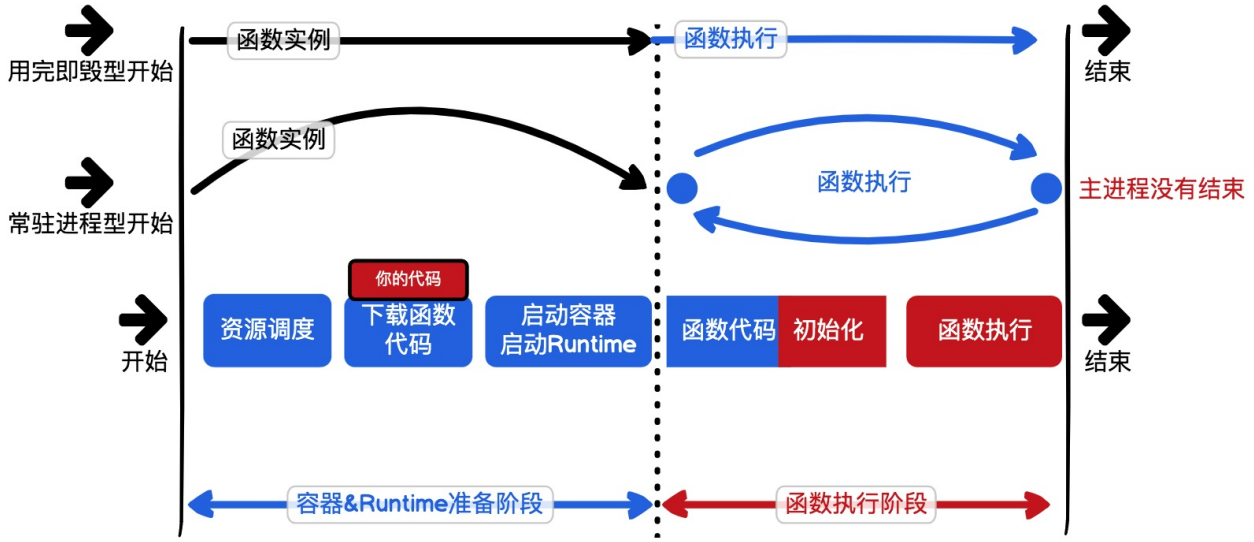
FaaS进程模型

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。**



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。**

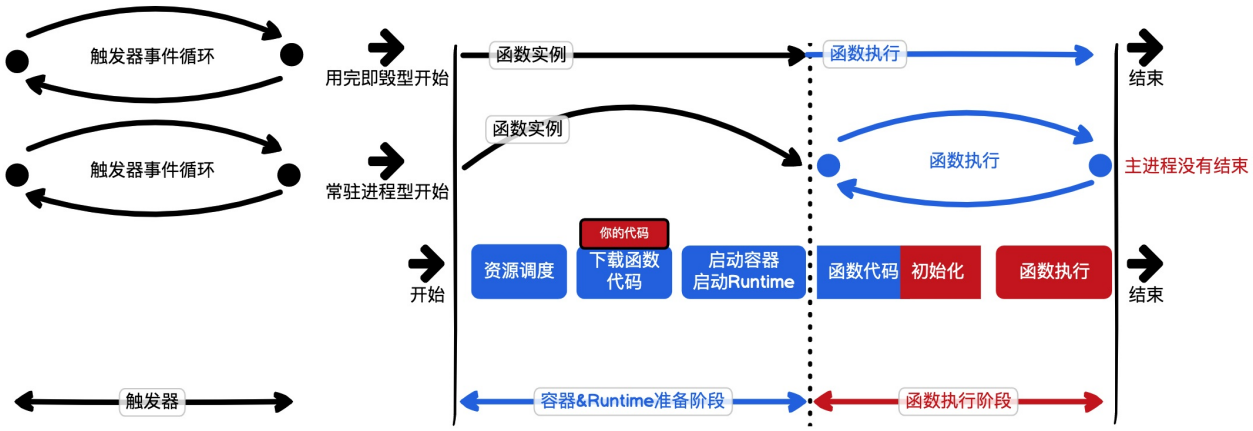
这里我要做个对比。在之前，假设没有FaaS，我们要将应用部署到托管平台PaaS上；启动Web服务时，主进程初始化连接MongoDB，初始化完成后，持续监听服务器的80端口，直到监听端口的句柄关闭或主进程接收到终止信号；当80端口和客户端建立完TCP链接，有HTTP请求过来，服务器就会将请求转发给Web服务的主进程，这时主进程会创建一个子进程来处理这个请求。

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。



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了，用完即毁型也不太适合传统MVC架构的改造，也是一种权宜之计，但这是FaaS最纯正的用法，肯定还是有它的用武之地的。

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。

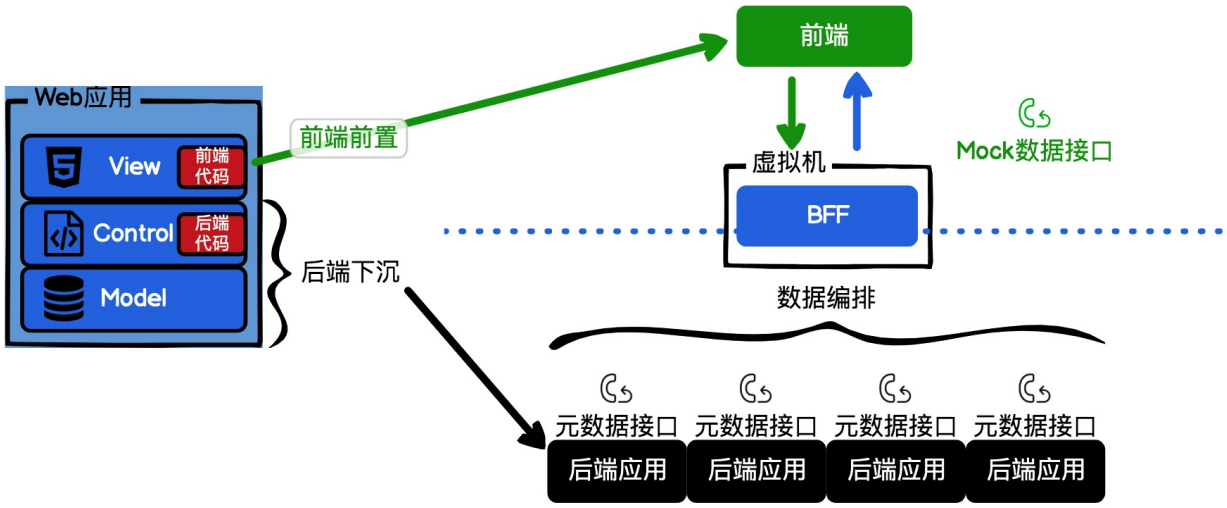
数据编排

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。

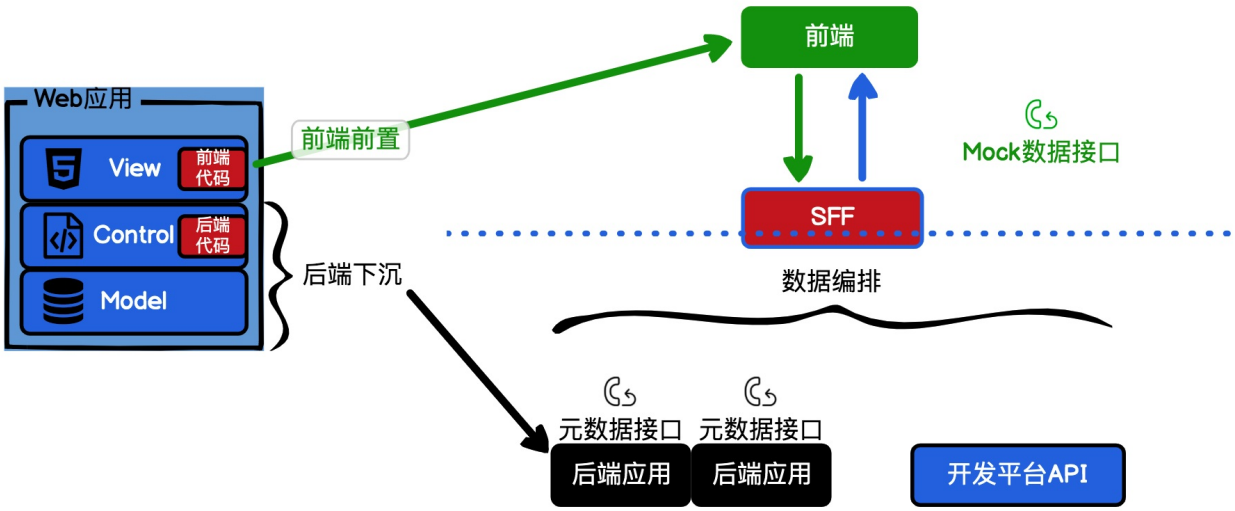


如上图所示，BFF层充当了中间胶水层的角色，粘合前后端。未经加工的数据，我们称为元数据Raw Data，对于普通用户来说元数据几乎不可读。所以我们需要将有用的数据组合起来，并且加工数据，让数据具备价值。对于数据的组合和加工，我们称之为**数据编排**。

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。



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。”

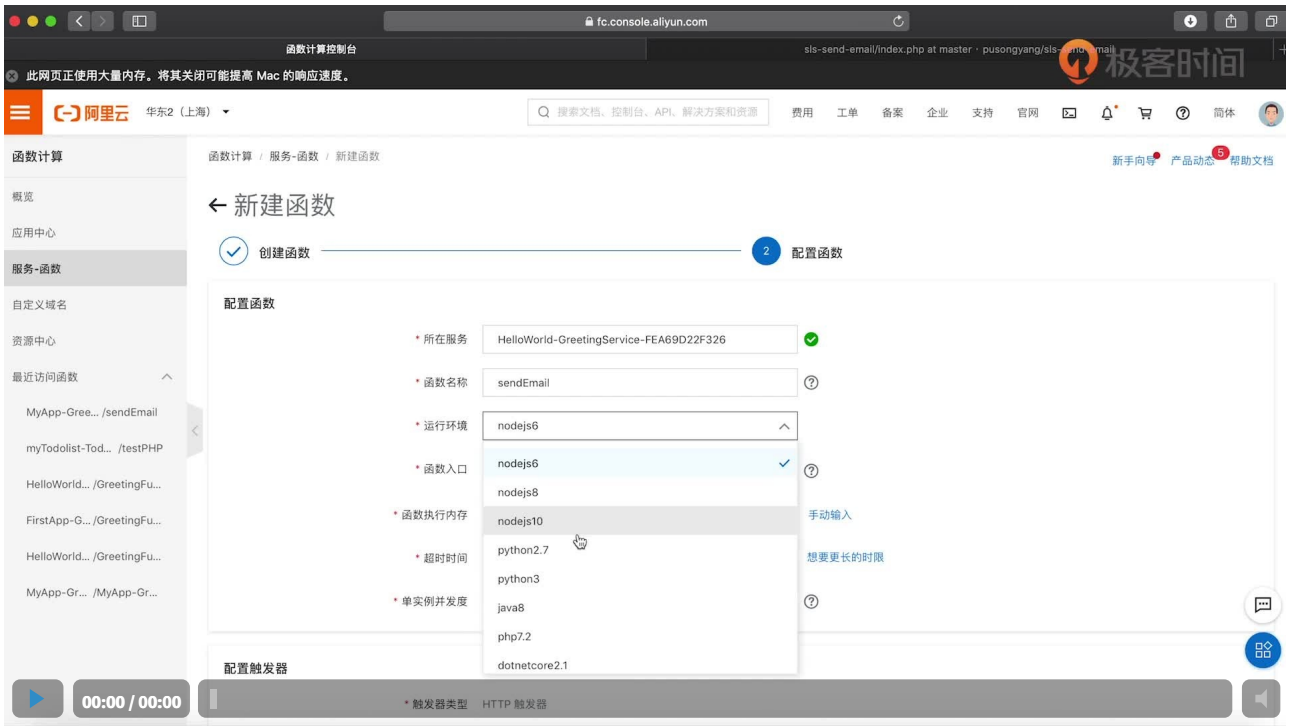
服务编排

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，费了没用的力气。

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，而且还与语言无关，这大大增加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/>

精选留言：

- 我来也 2020-04-22 12:12:56
  # 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在600-800ms.
  但是之后热启动时的耗时就只有30-50ms.
  而golang的冷启动时长不知道为什么要2.5s.而热启动后的接口耗时也才60ms.

  我还发现,阿里云上,5分钟以后,常驻进程模型的函数就被干掉了.症状就是初次接口耗时又要2s+.
  可以肯定的是,不到10分钟,无响应的函数就被系统回收了.肯定是没到15分钟.

我还发现,介于冷启动和热启动之间,还有一个状态,接口的响应耗时也是介于两者之间.

对服务编排的个人感悟
感觉函数服务配合服务编排,就像是在linux上使用shell组合各种命令,实现复杂的功能.
虽然每个命令都很简单,但是组合后的功能就很强大了.
现在的云服务都会有很多现成的sdk,确实如老师所说,需要用到某个云服务时临时把官方的文档拿出来,几乎只需要做很少的变动,就可以马上投入使用.

我目前使用函数服务,配合nas和日志服务,就可以很容易的把东西存在nas上,在阿里云的日志服务中搜索相关日志.
不足之处就是调试没有之前方便了.
[3赞]

作者回复2020-04-22 16:27:10
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。

- 一步 2020-04-22 10:11:57
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？ [2赞]

作者回复2020-04-22 16:29:40
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。

- 一步 2020-04-22 08:13:31
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？ [2赞]

作者回复2020-04-22 09:05:42
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。

- Bora.Don 2020-04-24 09:21:20
对老师关于数据编排的看法，存在一点疑问：数据不在本地，会不会存在稳定性和速度的问题？
FaaS感觉看上去很美，实际操作空间还有待考验，个人观点。

- Marooned。 2020-04-22 14:51:24
因为用完即毁型对传统 MVC 改造的成本太大 为什么会大呢？