## 01-定义：到底什么是Serverless?

你好，我是秦粤。Serverless目前是大热的话题，相信你肯定听过。但如果你去百度、Google或者维基百科上查的话，你会发现它连个准确的定义都没有。

作为本专栏的第一讲，今天我就想带你深入地了解下Serverless，看看它都能解决哪些问题，以及为什么难定义。

### Serverless能解决什么问题?

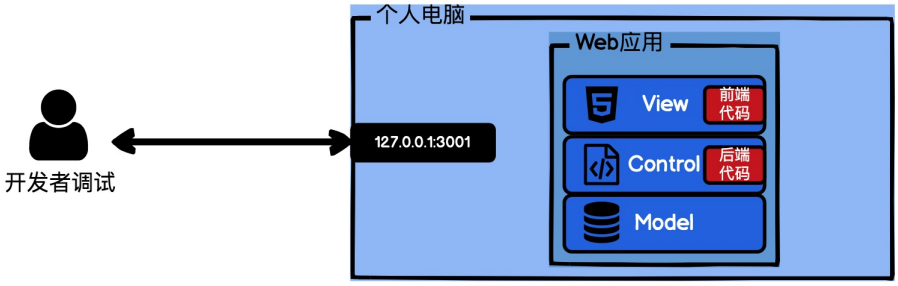
理清Serverless要解决的问题其实很简单，我们可以从字面上把它拆开来看。

Server这里指服务端，它是Serverless解决问题的边界；而less我们可以理解为较少关心，它是Serverless解决问题的目的。组合在一起就是“较少关心服务端”。怎么理解这句话呢？我们依然是拆开来分析。

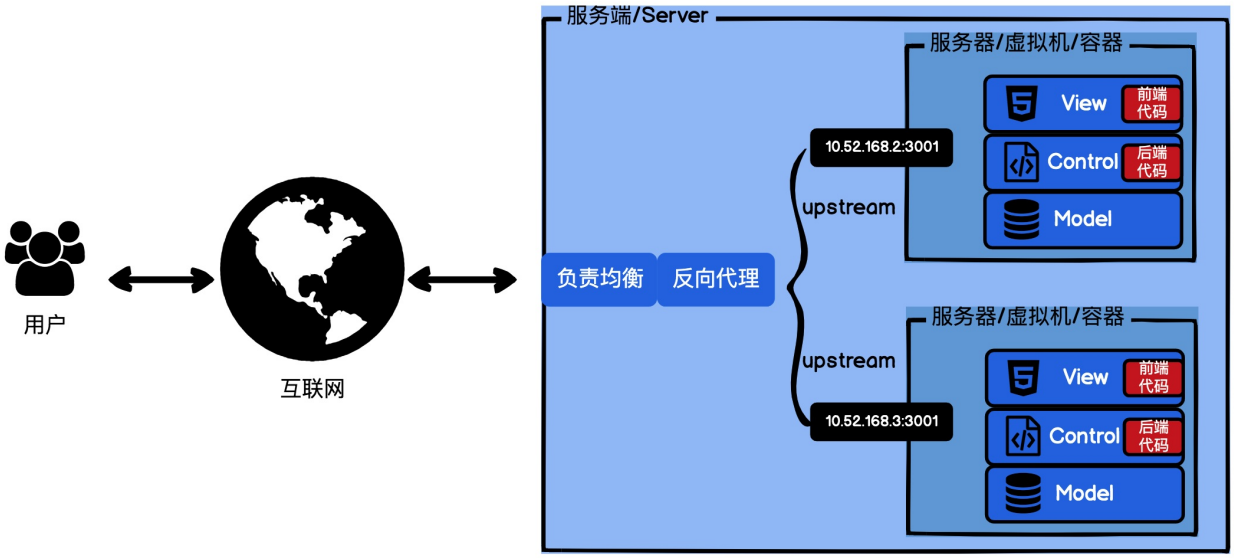
### 什么是服务端?

我们先看Server，这里我用Web应用经典的MVC架构来举例。

现代研发体系主要分为前端和后端，前端负责客户终端的体验，也就是View层；后端负责商业的业务逻辑和数据处理，也就是Control层和Model层。如果你有过一些开发经验，应该会了解自己的代码在本地开发和调试时的数据流。




通常我们会在自己电脑上启动一个端口号，例如127.0.0.1:3001。浏览器访问这个地址，就可以调用和调试自己的代码。但如果我们要将这个Web应用部署到互联网上，提供给互联网用户访问，就需要服务端的运维知识了。




通常我部署运维一个应用时，由于要考虑容灾和容错，我都会保障异地多活，因此我会部署多个Web应用实例。每个Web应用实例跟我们在本地开发时是一样的，只是IP改为了私有网络IP。

随着云服务商的兴起，现在已经很少有互联网企业还在自己维护物理机了。在云服务的运维体系中，各个环节都已经有了对应的成熟的云服务产品或解决方案。

为了使多个Web应用实例在容灾和容错的场景下稳定地切换流量，我就需要负载均衡服务和反向代理服务。负载均衡服务，正如其名是负责将流量均衡地分配到各个应用机器上。反向代理，常见的就是Nginx，它的任务是从请求中解析出域名信息，并将请求转发到上游upstream的监听地址。

服务端的边界，就是上图中的整个蓝色部分，它是负责应用或代码的线上运维。Serverless解决问题的边界，就是服务端的边界，即服务端运维。

### 服务端运维发展史，从full到less

了解完Server，我们再来看less。

我们可以先看Serverfull的概念，对比Serverfull和Serverless之间的差别，相信这样可以加深你的理解。Serverfull就是服务端运维全由我们自己负责，Serverless则是服务端运维较少由我们自己负责，大多数的运维工作交给自动化工具负责。

可能这么说比较抽象，我举个例子来给你讲吧。这个例子有点长，但可以带你很好地了解下服务端运维的发展史，我们后面也会再次用到。

假设我有一家互联网公司，我的产品是“待办任务（ToDoList）”Web应用——记录管理每个用户的待办任务列表。针对这个网站的研发，我们简化为两个独立角色：研发工程师小程和运维工程师小服。

做研发的小程，他是个精通前后端的全栈工程师，但他只关心应用的业务逻辑。具体来说就是，整个MVC架构Web应用的开发都归小程负责，从客户端界面View层，到业务逻辑Control层，再到数据存储Model层，整个Web应用的版本管理和线上bug修复。

负责运维的小服，则只关心应用的服务端运维事务。他负责部署上线小程的Web应用，绑定域名以及日志监控。在用户访问量大的时候，他要给这个应用扩容；在用户访问量小的时候，他要给这个应用缩容；在服务

器挂了的时候，他还要重启或者换一台服务器。

**史前时代，Serverfull**

最开始运维工程师小服承诺将运维的事情全包了，小程不用关心任何部署运维相关的事情。小程每次发布新的应用，都会打电话给小服，让小服部署上线最新的代码。小服要管理好迭代版本的发布，分支合并，将应用上线，遇到问题回滚。如果线上出了故障，还要抓取线上的日志发给小程解决。

小程和小服通过工作职责任务上的安排，将研发和运维完全隔离开来了。好处很明显：分工明确小程可以专心做好自己的业务。缺陷也很明显：小服成了工具人，被困在了大量的运维工作中，处理各种发布相关琐碎的杂事。

这个时代研发和运维隔离，服务端运维都交给小服一个人，纯人力处理，也就是Serverfull。

我们可以停下来想想，像发布版本和处理线上故障这种工作这些是小程的职责，都要小服协助，是不是应该小程自己处理?

**农耕时代，DevOps**

后来，小服渐渐发现日常其实有很多事情都是重复性的工作，尤其是发布新版本的时候，与其每次都等小程电话，线上出故障了还要自己抓日志发过去，效率很低，不如干脆自己做一套运维控制台OpsConsole，将部署上线和日志抓取的工作让小程自己处理。

OpsConsole上线后，小服稍微轻松了一些，但是优化架构节省资源和扩缩容资源方案，还是需要小服定期审查。而小程除了开发的任务，每次发布新版本或解决线上故障，都要自己到OpsConsole平台上去处理。

这个时代就是研发兼运维DevOps，小程兼任了小服的部分工作。小服将部分服务端运维的工作工具化了，自己可以去做更加专业的事情。相对史前时代，小程负责的更多，看起来是不是小程负责的事情多（More）了？但实际这些事情本身就应该是小程负责的。版本控制、线上故障都是小程自己应该处理的。而且小服将这部分人力的工作工具化了，更加高效。其实已经有变少（less）的趋势了。

我们再想想能否进一步提升效率，让小程连OpsConsole平台都可以不用?

**工业时代**

这时，小服发现资源优化和扩缩容方案也可以利用性能监控+流量估算解决。小服又基于小程的开发流程，OpsConsole系统再进一步，帮小程做了一套代码自动化发布的流水线：代码扫描-测试-灰度验证-上线。现在的小程连OpsConsole都不用登陆操作，只要将最新的代码合并到Git仓库指定的develop分支，剩下的就都由流水线自动化处理发布上线了。

这个时代研发不需要运维了，免运维NoOps。小服的服务端运维工作全部自动化了。小程也变回到最初，只需要关心自己的应用业务就可以了。我们不难看出，在服务端运维的发展历史中，对于小服来说，小程的角色存在感越来越弱，需要小程参与的事情越来越少，都由自动化工具替代了。这就是“Serverless”。

到这里你一定会想，既然服务端都做到免运维了，小服是不是就自己革了自己的命，失业了?

**未来**

实现了免运维NoOps，并不意味着小服要失业了，而是小服要转型。转型去做更底层的服务，做基础架构的建设，提供更加智能、更加节省资源、更加周到的服务。小程则可以完全不被运维的事情困扰，放心大胆地依赖Serverless服务，专注做好自己的业务，提升用户体验，思考业务价值。

免运维NoOps并不是说服务端运维就不存在了，而是通过全知全能的服务，覆盖研发部署需要的所有需求，让研发同学小程对它的感知越来越少。另外，NoOps是理想状态，因为我们只能无限逼近NoOps，所以这个单词是less，不可能是Server**Least**或者Server**Zero**。

另外你需要知道的是，目前大多数互联网公司，包括一线互联网公司，都还在DevOps时代。但Serverless的概念已经提出，NoOps的时代正在到来。

Server限定了Serverless解决问题的边界，即服务端运维；less说明了Serverless解决问题的目的，即免运维NoOps。所以我们重新组合一下这个词的话，Serverless就应该叫做服务端免运维。这也就是Serverless要解决的问题。

## Serverless为什么难准确定义?

换句话说，服务端免运维，要解决的就是将小服的工作彻底透明化。研发同学只关心业务逻辑，不用关心部署运维和线上的各种问题。要实现这个终态，就意味着需要对整个互联网服务端的运维工作进行极端抽象。

那越抽象的东西，其实越难定义，因为蕴含的信息量太大了，这就是Serverless很难准确定义的根本原因。Serverless对Web服务开发的革命之一，就是极度简化了服务端运维模型，使一个零经验的新手，也能快速搭建一套低成本、高性能的服务端应用（下一讲，我就会带你体验一下从零开始的Serverless应用）。

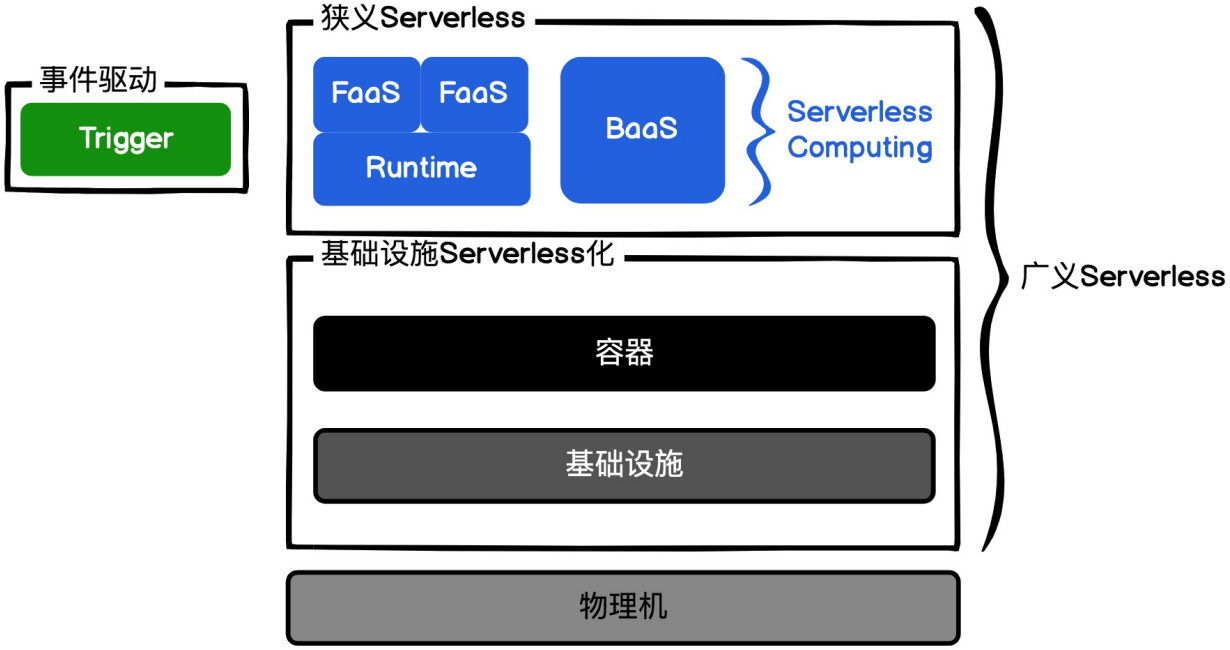
我们现在虽然知道了Serverless的终态是NoOps，但它作为一门新兴的技术，我们还是要尝试给它定义吧?

## 到底什么是Serverless?

我在日常和其他同事沟通的时候，我发现大家对Serverless概念的认知很模糊，往往要根据沟通时的上下内容去看到底此时的Serverless在指代什么。主要是因为Serverless这个词包含的信息量太大，而且适用性很广，但总结来说Serverless的含义有这样两种。

第一种：狭义Serverless（最常见）= Serverless computing架构 = FaaS架构 = Trigger（事件驱动）+ FaaS（函数即服务）+ BaaS（后端即服务，持久化或第三方服务）= FaaS + BaaS

第二种：广义Serverless = 服务端免运维 = 具备Serverless特性的云服务




我用图片来阐明一下这两种概念。其实你不难看出，广义Serverless包含的东西更多，适用范围更广，但我们经常在工作中提到的Serverless一般都是指狭义的Serverless。其实这是历史原因，2014 年11月份，亚马逊推出真正意义上的第一款Serverless FaaS服务：Lambda。Serverless的概念才进入了大多数人的视野，也因此Serverless曾经一度就等于FaaS。

我们再聚焦到图上“狭义Serverless”这里。你注意看的话，图中有几个陌生的名词，我先来给你解释下。FaaS(Function as a Service)就是函数即服务，BaaS(Backend as a Service)就是后端即服务。XaaS(X as a Service)就是X即服务，这是云服务商喜欢使用的一种命名方式，比如我们熟悉的SaaS、PaaS、IaaS都是这样。

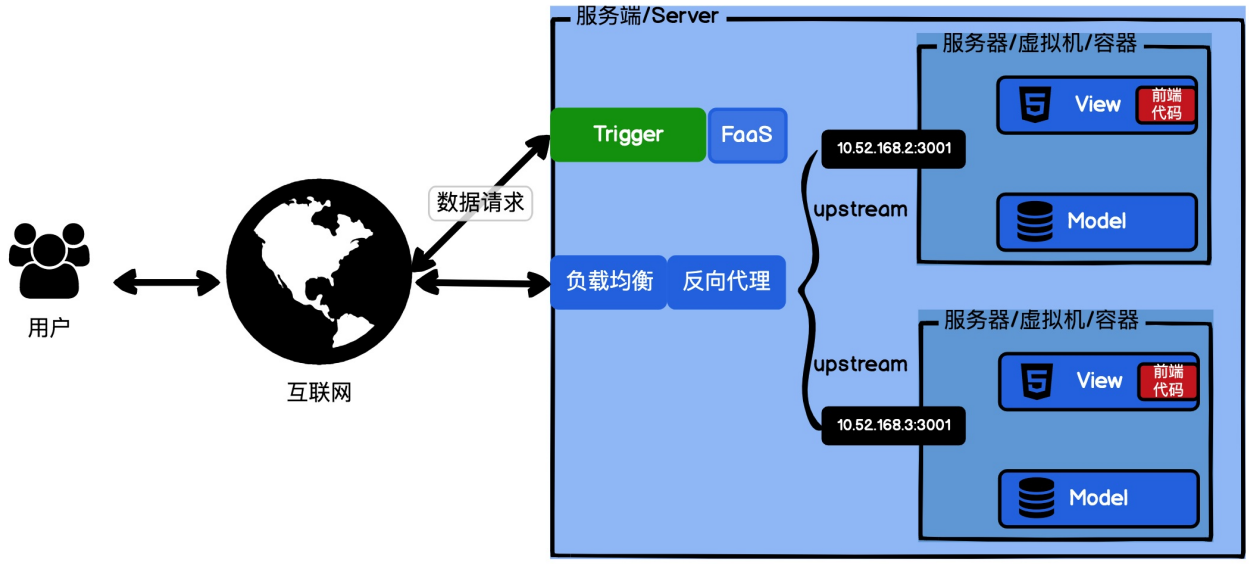
先说FaaS，函数即服务，它还有个名字叫作Serverless Computing，它可以让我们随时随地创建、使用、销毁一个函数。

你可以想一下通常函数的使用过程：它需要先从代码加载到内存，也就是实例化，然后被其它函数调用时执行。在FaaS中也是一样的，函数需要实例化，然后被触发器Trigger或者被其他的函数调用。二者最大的区别就是在Runtime，也就是函数的上下文，函数执行时的语境。

FaaS的Runtime是预先设置好的，Runtime里面加载的函数和资源都是云服务商提供的，我们可以使用却无法控制。你可以理解为FaaS的Runtime是临时的，函数调用完后，这个临时Runtime和函数一起销毁。

FaaS的函数调用完后，云服务商会销毁实例，回收资源，所以FaaS推荐无状态的函数。如果你是一位前端工程师的话，可能很好理解，就是函数不可改变Immutable。简单解释一下，就是说一个函数只要参数固定，返回的结果也必须是固定的。

用我们上面的MVC架构的Web应用举例，View层是客户端展现的内容，通常并不需要函数算力。Control层，就是函数的典型使用场景。MVC架构里面，一个HTTP的数据请求，就会对应一个Control函数，我们完全可以用FaaS函数来代替Control函数。在HTTP的数据请求量大的时候，FaaS函数会自动扩容多实例同时运行；在HTTP的数据请求量小时，又会自动缩容；当没有HTTP数据请求时，还会缩容到0实例，节省开支。

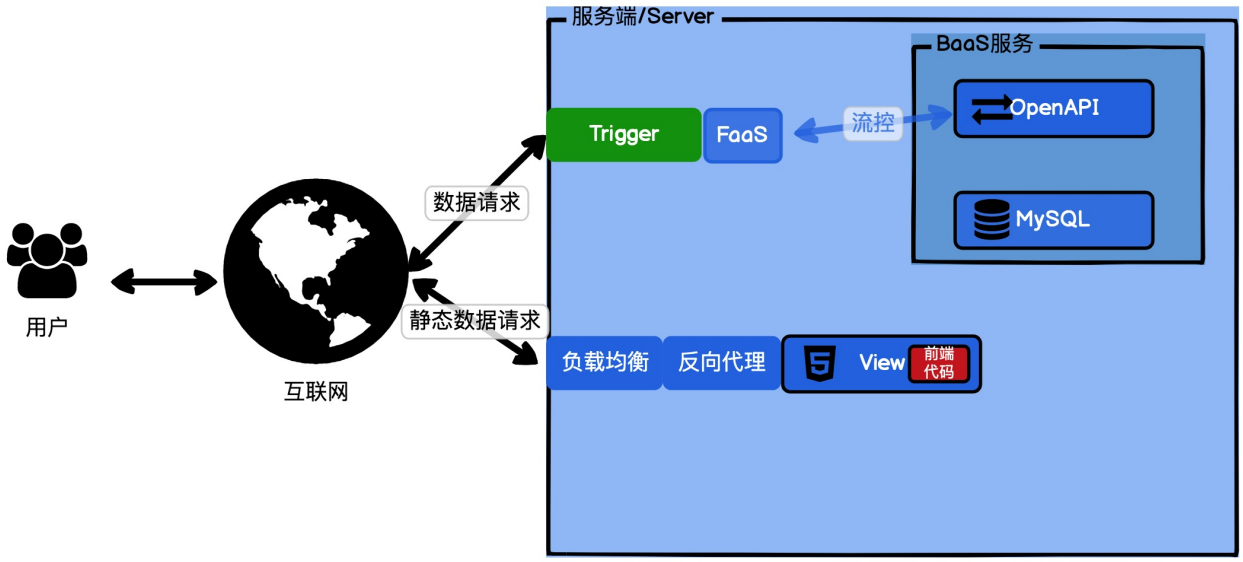



此刻或许你会有点疑惑，Runtime不可控，FaaS函数无状态，函数的实例又不停地扩容缩容，那我需要持久化存储一些数据怎么办，MVC里面的Model层怎么解决？

此时我就要介绍另一位嘉宾，BaaS了。

BaaS其实是一个集合，是指具备高可用性和弹性，而且免运维的后端服务。说简单点，就是专门支撑FaaS的服务。FaaS就像高铁的车头，如果我们的后端服务还是老旧的绿皮火车车厢，那肯定是要散架的。而BaaS就是专门为FaaS准备的高铁车厢。

MVC架构中的Model层，就需要我们用BaaS来解决。Model层我们以MySQL为例，后端服务最好是将FaaS操作的数据库的命令，封装成HTTP的OpenAPI，提供给FaaS调用，自己控制这个API的请求频率以及限流降级。这个后端服务本身则可以通过连接池、MySQL集群等方式去优化。各大云服务商自身也在改造自己的后端服务，BaaS这个集合也在日渐壮大。




基于Serverless架构，我们完全可以把传统的MVC架构转换为BaaS+View+FaaS的组合，重构或实现。

这样看下来的话，狭义Serverless的含义也就不难理解了。

> 第一种：狭义Serverless（最常见）= Serverless computing架构 = FaaS架构 = Trigger（事

> 件驱动）+ FaaS（函数即服务）+ BaaS（后端即服务，持久化或第三方服务）= FaaS + BaaS

Serverless毋庸置疑正是因为FaaS架构才流行起来，进入大家认知的。所以我们最常见的Serverless都是指Serverless Computing架构，也就是由Trigger、FaaS和BaaS架构组成的应用。这也是我给出的狭义Serverless的定义。

**那什么是广义Serverless呢?**

将狭义的Serverless推升至广义，具备以下特性的，就是Serverless服务。你可以回忆一下小服的工作，要达成NoOps，都应该具备什么条件?

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）。

广义Serverless，其实就是指服务端免运维，也是未来的主要趋势。

总结来说的话就是，我们日常谈Serverless的时候，基本都是指狭义的Serverless，但当我们提到某个服务Serverless化的时候，往往都是指广义的Serverless。我们后面的课程中也是如此。

## 总结

今天我们一起学习了Serverless的边界、目标以及定义。我还介绍了即将贯穿我们整个专栏的Web应用，即任务列表。

现在我们可以回过头来看看最初的两个问题了，你是不是已经有答案了呢?

1. Serverless能解决什么问题？Serverless可以使应用在服务端免运维。
2. Serverless为什么难定义？Serverless将服务端运维高度抽象成了一种解决方案，包含的信息量太大了。

另外，Serverless可分为狭义和广义。狭义Serverless是指用FaaS+BaaS这种Serverless架构开发的应用；广义Serverless则是具备Serverless特性的云服务。现在的云服务商，正在积极地将自己提供的各种云服务Serverless化。

## 作业

最后，留给你一个作业：请你注册一个云服务商的云账号，并根据云服务商的文档和教程，自己部署一个Serverless应用。下节课，我将用云上的FaaS部署一个Serverless应用，给你详细讲解Serverless引擎盖内的知识。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。

## 精选留言：

- 罗祥 2020-04-16 08:08:46
  Serverless 让服务端免运维，这一点很赞，发布应用不需要运维了。
  老师留的作业准备今天中午实践一下，实践完再来留言。 [1赞]

作者回复2020-04-16 09:14:12
嗯，Serverless需要大家自己多体验一下，才能有切身感受。

- 唔多志 2020-04-15 23:23:38
老师的思路还是很赞，对 Serverless 有了比较清晰饿认识。 [1赞]

作者回复2020-04-16 09:16:16
多谢你的支持！

- pedro 2020-04-15 18:38:12
这里不得不提微信小程序的云开发其实就是一种意义上的 serverless，让前端工程师不仅可以开发页面还可以通过云函数(Faas)来写业务，而且还提供了基础存储(Baas)。 [1赞]

作者回复2020-04-15 20:35:56
是的，微信小程序云开发，也是一种serverless的应用场景。Serverless发展的一个方向，也在追求这种一体化的开发体验。

- 每天晒白牙 2020-04-16 06:47:09
这样搞下去，程序员门槛越来越低，如果不提高自己的核心竞争力，真的要失业了😂

作者回复2020-04-16 09:15:06
不用担心，程序员跟其他职业一样只不过会往更专业，更精细化的分工去走。

- David.kor 2020-04-15 23:02:24
FaaS 在aws上可以通过API Gateway+lambda实现，请问老师BaaS具体可以通过什么方式实现呢？

作者回复2020-04-16 09:17:29
同学你能这样问说明你在认真学习和思考，赞一个！我专栏后续的课程会将，怎么将后端的应用BaaS化。当然现在云服务商也提供了很多BaaS的能力。

- JackPn 2020-04-15 19:08:58
这个有点牛皮，感觉这样一来程序员的技术活只剩下写逻辑了，其他的都是管理

作者回复2020-04-15 23:48:20
Serverless还在发展阶段，体验也还在完善中，但肯定是未来值得关注的内容。未来技术门槛只会越来越低。程序员和其他的工作一样，应该是去拼想象力，并不是除了技术就是管理。