

34 | GitOps 开发循环慢，时间都去哪了？

2023-02-24 王伟 来自北京

《云原生架构与GitOps实战》

[课程介绍 >](#)



讲述：王伟

时长 11:08 大小 10.16M



你好，我是王伟。

从这节课开始，我们正式进入到云原生开发领域的学习。

相比较传统的单体应用，云原生应用有非常多的不同之处。比如，云原生应用往往由多个微服务组成，使用了容器技术解决业务的打包和运行问题，此外，它还使用了很多云上的托管服务，例如 MQ，数据库等。

随着业务应用对云原生的依赖越来越大，我们会逐渐发现云原生应用的开发越来越复杂，也变得越来越慢。你可以把这理解为把应用迁移到云原生架构下的“副作用”，在今天，这些问题仍然没有最优的解决方案。

这节课，我会带你从几个角度了解造成云原生开发变慢的原因，从开发循环反馈的概念讲到架构演进。此外，我还会比较单体应用和云原生应用在开发上差异，带你深入理解为什么会产生

这些“副作用”。最后介绍两种提升开发效率的技术方案。

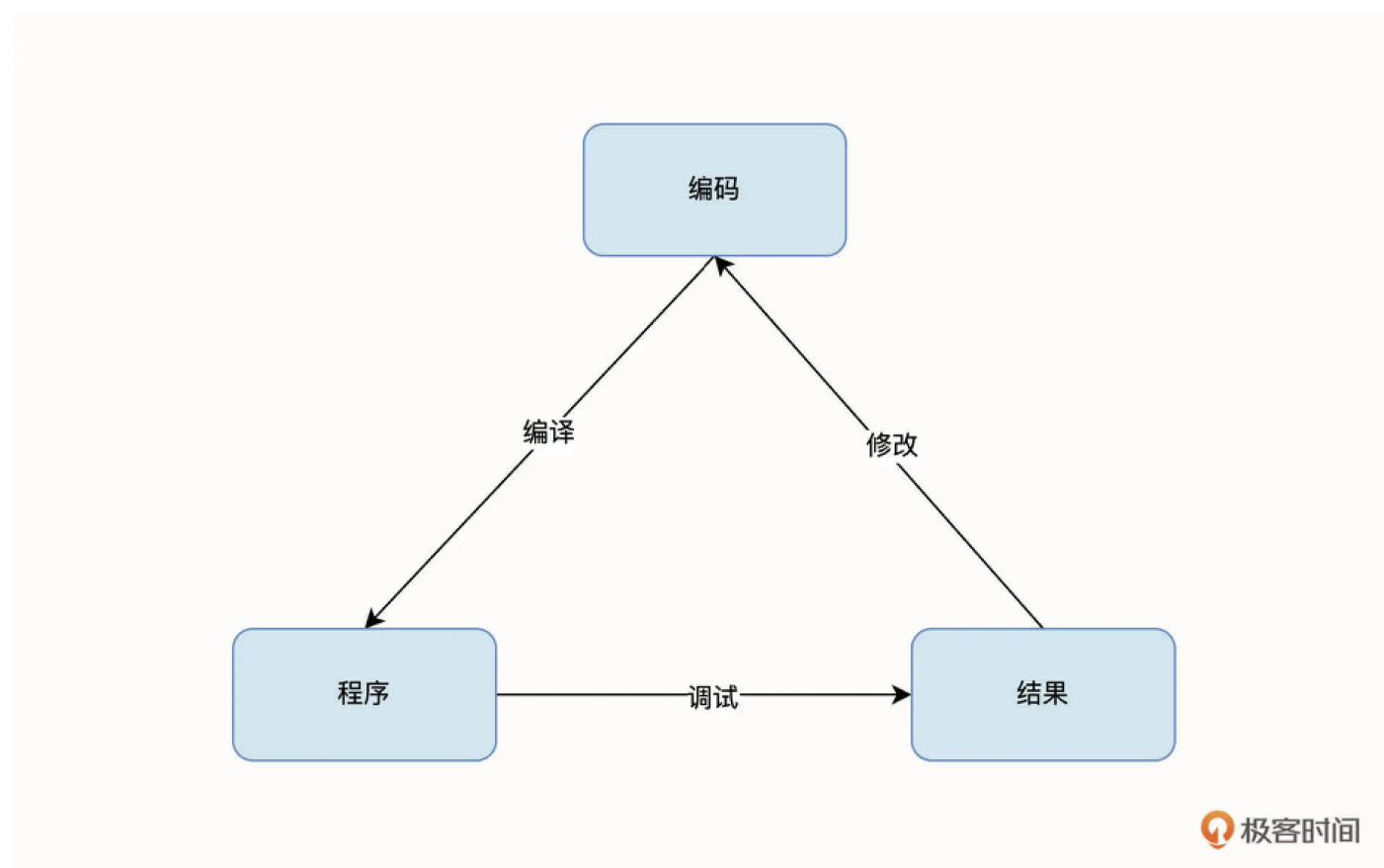
开发循环反馈

我们先来了解一下什么是开发循环反馈，这要从软件研发过程开始说起。

在软件研发的过程中，开发者在接到需求后会进行构思、编码。根据编码难度的不同，在编写完一段代码后，它们需要对程序进行编译并运行，以便查看结果。如果结果不符合预期，则会尝试输出一些变量值进行调试，如果这样不能满足调试需求，就需要进行断点调试。

调试完成并找到问题后，再修复代码，并重新进行上述的过程。

这个完整的过程，也就是我们说的开发循环反馈，如下图所示。



通过上面的描述，我们很容易得出下面几个结论。

1. 大部分情况下，一次开发循环反馈验证的代码量在几行和几十行之间，取决于编码难度。
2. 在完成一个需求时，我们可能需要经历数十个甚至数百个开发循环反馈。
3. 开发循环反馈的效率很大程度上决定开发效率。

在上面的结论中，第一点和第二点都很好理解。关于第三点我举一个例子，在开发一些大型的 **Java** 项目的时候，修改代码后要查看编码效果，编译和启动过程可能长达 **10** 分钟。这意味着，每次编码循环反馈至少有 **10** 分钟是花在无效的等待上的。也就是说，满打满算一小时只能进行 **6** 次开发循环反馈，**这是导致开发效率变低的原因之一**。不过显然，这并不是将应用迁移到云原生架构导致的。

开发循环反馈在架构演进下的差异

现在，让我们来简单回顾下在不同的架构下开发循环反馈的差异。

我将按照架构演进顺序，依次分析**前云时代架构**、**云时代架构**和**云原生时代架构**。

前云时代架构

在前云时代架构下，我们通常使用瀑布的开发方式，最经典的例子是操作系统。通常，我们需要数年的时间进行研发，也不经常发布更新。

在这种架构体系下，**开发循环反馈的效率取决于项目的大小，但通常我们可以很容易地在本地进行开发。**

云时代架构

在云时代架构下，敏捷和 **DevOps** 的开发方法代替了瀑布的开发方法，我们可以更快地开发和交付应用。

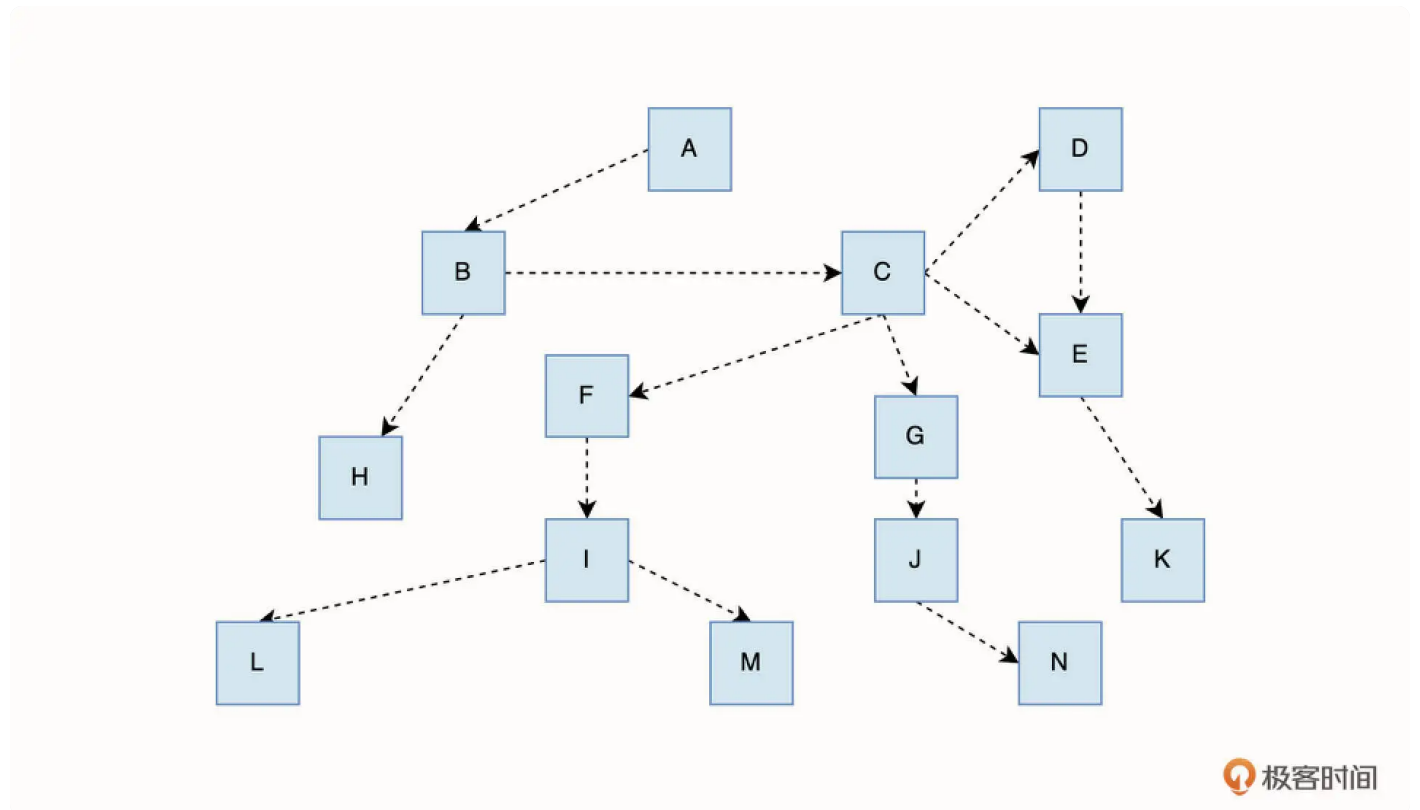
相比较前云时代的架构，云时代的架构更多使用了云端的技术，例如远程和云上开发开始出现，我们可以通过连接云端的机器进行开发和编译，这在一定程度解决了本地资源不足导致的编译和启动过慢的问题。

在这种架构体系下，我们可以使用**大型的云上开发机**来提升开发循环反馈效率，这种开发方式逐渐变得流行起来。

云原生时代架构

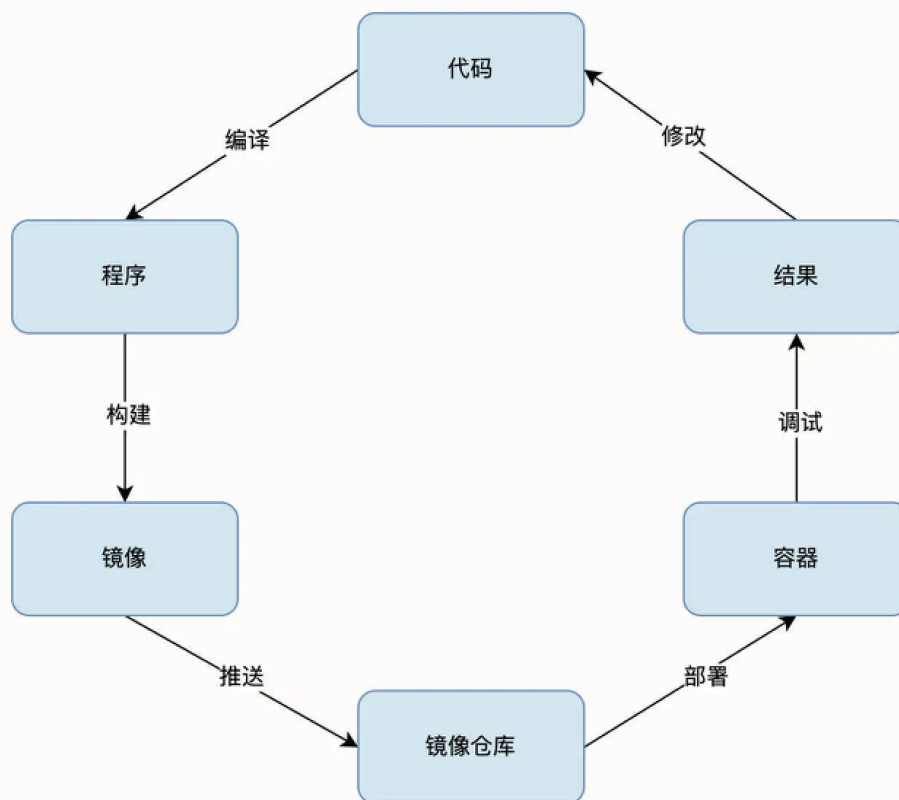
在云原生时代架构下，业务应用产生了巨大的变化：**微服务**。

越来越多的单体应用被拆分成了微服务，微服务各司其职，通常一个完整的业务流程需要调用多个微服务才能完成，如下图所示。



在微服务架构下，如果本地资源充足，只需要将所有依赖的微服务在本地运行起来，小型的应用仍然可以在本地进行开发。待开发的微服务仍然可以像单体应用一样进行编码、编译和调试。在这种场景下，**开发循环反馈没有出现变化**。

但是，随着容器化改造和迁移到 Kubernetes，尤其是对于那些无法在本地开发的大型业务应用来说，原有的开发循环反馈流程发生了**巨大的变化**。



从上面这张图我们可以看出，在非容器化架构下，开发循环反馈流程只有编码、编译和调试过程。但在将应用迁移到容器和 Kubernetes 架构时，要查看编码效果，我们必须要先**构建镜像、推送到镜像仓库并等待容器重启**才能看到结果，开发效率也随之大幅降低。

我们已经知道，微服务和容器化是造成开发循环反馈产生变化的最根本原因。那么，要怎么理解它们呢？

首先，业务应用在进行微服务改造之后，由于不同应用的环境差异，依赖差异以及本地的资源问题，在开发过程，我们很难在本地将所有微服务都启动起来。在这种情况下，使用**云端的 Kubernetes 集群**作为开发环境是唯一的解决办法。

此外，当业务进行容器化改造以及迁移到 Kubernetes 之后，当要查看某个微服务编码效果时，已经无法像单体应用一样只需要简单的编译和启动了，这是因为这个微服务可能需要依赖其他的微服务、数据库和中间件才能正常工作。这就导致我们只能将应用构建成镜像，并将它部署到**具备业务应用完整依赖**的远端 Kubernetes 集群，然后才能看到编码效果。

如果从编程的角度来理解，单体应用的调用关系是代码中的函数和类，而微服务应用的调用关系则是通过 HTTP 或 RPC 协议调用其他的微服务。

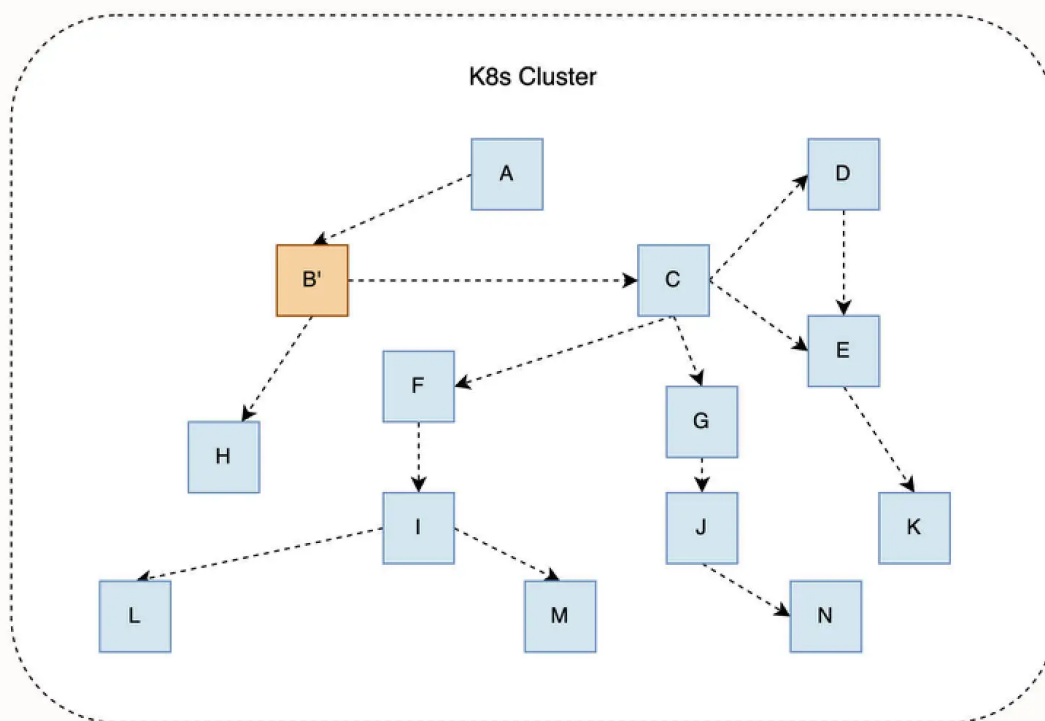
GitOps 架构下的开发循环反馈

通过上面的分析，我们已经大体上了解了为什么在 GitOps 架构下开发循环反馈的效率会变低。

总结来说，在 GitOps 架构下，如果我们要查看编码效果，我们会在下面这些阶段花费大量无效的等待时间：

1. 构建镜像
2. 推送到镜像仓库
3. 修改工作负载镜像版本并等待 Kubernetes 拉取镜像
4. 等待新镜像启动完成

以下面这张示意图为例，试想一下如果所有微服务都部署在云端的 Kubernetes 集群内，要将 B 服务修改为新镜像版本，是不是一定会经历上面的这些步骤呢？



到这里，我相信有一些同学会产生疑问，上面的这些阶段难道不是 GitOps 自动化的优势吗？为什么会变成拖累开发效率的原因？

从发布角度来说，这些步骤是必不可少的，这也是 GitOps 的优势。但从开发角度来说，每次编码循环反馈都需要经历这些步骤是**非常浪费时间的**。

在开发过程中，既然最大的耗时是在构建镜像的过程里，那么，怎样才能降低镜像构建对开发效率的影响呢？

提高开发循环反馈效率

在 GitOps 架构下，要提高开发循环反馈效率，我为你提供下面三个思路。

1. 提升镜像构建和拉取速度。
2. 本地 + 远程的混合开发方式。
3. 远程开发。

提升镜像构建和拉取速度

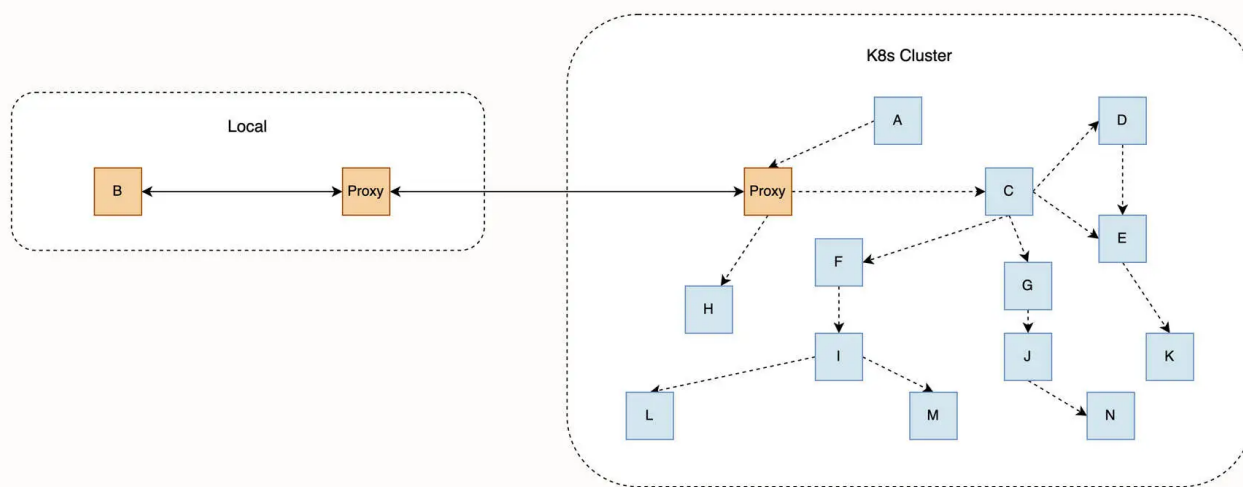
要提高开发循环的反馈效率，最容易想到的就是提升镜像构建和拉取速度。例如，你可以按照 [🔗 第 14 讲](#) 的内容减小镜像体积大小来加快构建速度，也可以优化 Dockerfile 让构建过程使用缓存来加快速度，还可以部署私有镜像仓库并将它和 Kubernetes 集群配置在同一个 VPC 网络下提升拉取速度。

这种方式虽然能在一定程度提高开发循环反馈效率，但由于它仍然需要构建镜像，所以并不能从根本上解决问题。

本地 + 远程的混合开发方式

我们已经知道，之所以需要将镜像部署到远端 Kubernetes 集群，是因为远端集群具备待开发微服务的所有依赖，例如数据库、中间件和其他微服务等等。

那换一个思路，如果我们可以将待开发的微服务在本地直接运行，是不是就不需要构建镜像了呢？如下图所示。



以开发 **B** 服务为例，要实现这种混合的开发方式，我们需要将集群的 **B** 服务变成 **Proxy** 代理，它负责将集群内对 **B** 服务的访问流量转发到本地，并在本地同时启动 **Proxy** 和 **B** 服务，这样，就可以将本地的 **B** 服务对集群的依赖服务（例如数据库、中间件和其他微服务）的请求代理到集群内了。

通过这种“双向代理”能力，我们就可以在本地以源码的方式开发 B 服务了。像开发单体应用一样，我们在编码后不再需要构建镜像等一系列操作，而是可以直接编译源码并启动，开发循环反馈回到了效率最高的时代。

不过，在真正要实现这种开发方式的时候，我们通常会遇到两个比较大的限制。

首先，在一些严格的内网环境下，Proxy 代理可能改变网络拓扑结构，原有网络可能失效，此外，Proxy 在一些场景下也可能无法全流量代理。

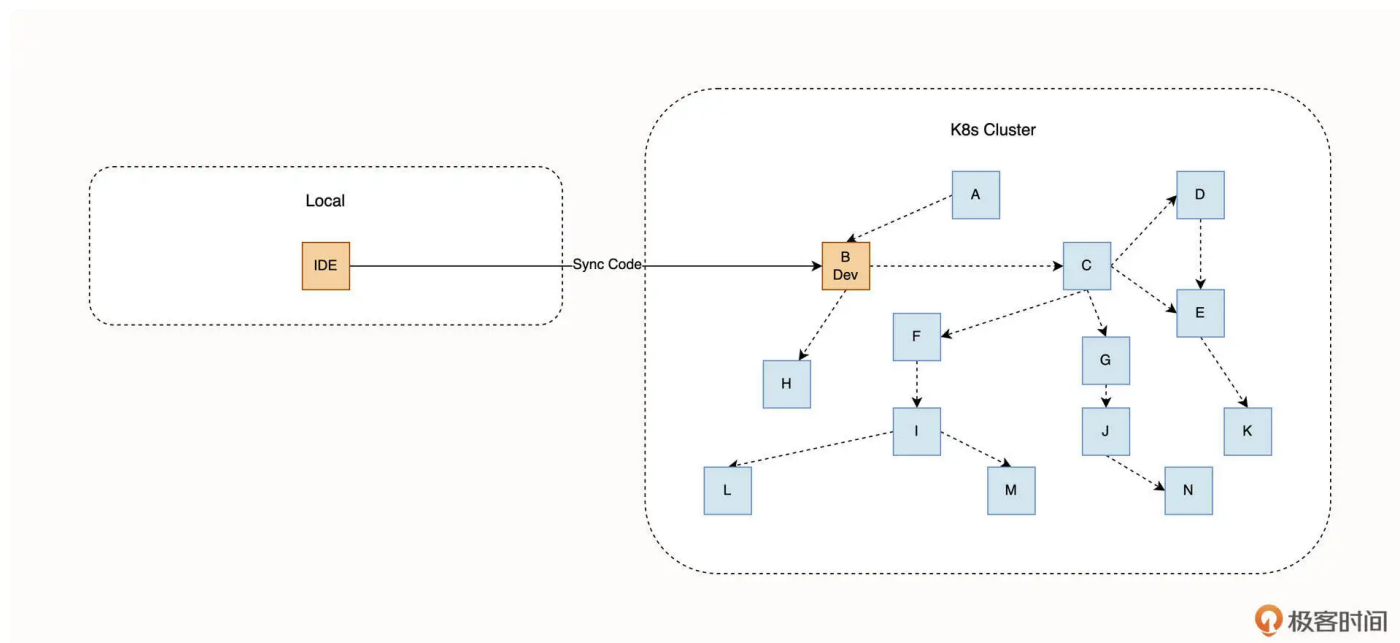
其次，要在本地启动 B 服务并不容易。在 Kubernetes 环境下，B 服务的配置可能是由 ConfigMap 或 Secret 提供的。但在本地我们并不能模拟这两种类型的配置文件，在配置非常复杂的情况下，手动编写配置也比较困难。

所以，这种方式比较适合业务应用比较小，并且某个微服务可以很轻易地在本地运行的情况。

远程开发

最后，我再来介绍一种远程开发方式，它可以很好地解决我们在上面提到的一系列的限制。

远程开发核心的思想仍然是：**复用远端 Kubernetes 集群的环境和依赖**，如下图所示。



以开发 B 服务为例，和本地 + 远程的混合开发方式不同的是，远端开发的核心原理是将 B 服务修改为“开发模式”，让其以源码的方式启动，并将本地源码的实时修改同步到远端集群 B 服务的容器内，通过重启业务进程的方式，实现本地编码实时生效的效果。

由于这种方式不需要在本地启动微服务，所以本地只需要有 IDE 编辑器即可，不需要任何编程环境。此外，由于业务进程仍然是在原来的容器内启动的，所以配置文件、ConfigMap 和 Secret 的读取方式没有产生任何变化，这也是业务进程能够在容器里直接启动的根本原因，也是实现远程开发的基础。

在进行编码时，远程开发方式不需要重新构建镜像，开发体验就和在本地开发单体应用一样，只需要经过编译和启动即可。

不过，值得注意的是，由于编译和启动过程都是在容器内进行的，所以通常我们需要为容器设置较大的资源配额，以便获得更好的开发体验。

从通用性和体验角度来看，在 GitOps 架构下，远程开发是我最推荐的开发方式。

总结

这节课，我们介绍了开发循环反馈的概念以及它在不同架构演进下的差异，尤其是在云原生时代，随着微服务和容器技术的采用，开发循环反馈发生了巨大的变化。

在简单的单体应用场景下，开发循环反馈只需要经历编码、编译和运行过程。但在云原生时代，这种极简的编程体验已经不复存在了。

产生这种巨大变化的根本原因是：受到资源限制，大型云原生微服务应用已经很难在本地进行开发，我们不得不借助云端 **Kubernetes** 集群作为开发环境，这就导致要查看编码效果，就必须经历构建镜像、推送到镜像仓库、修改工作负载的镜像版本并等待新镜像启动的过程，而这些过程正是开发效率变低的重要因素，开发循环反馈从单体应用的秒级变成了分钟级。

对于云原生开发来说，目前社区并没有标准的解决方案。对此，我介绍了三种解决方案，它们分别是提升镜像构建和拉取速度、本地 + 远程混合的开发方式以及远端开发的方式。

其中，提升镜像构建和拉取速度并不能从根本上解决问题，本地 + 远程混合开发方式虽然改变了开发循环反馈的方式，但它受限于使用场景且不够通用，而远端开发方式则是一种通用性更强的开发方式，也是我将在下一节课重点介绍的内容。

下一节课，我将带你深入远程开发，并带你学习如何使用 **CNCF Sandbox** 项目 **Nocalhost** 来开发 **Kubernetes** 应用。


思考题

最后，给你留一道思考题吧。

你能和我们分享你现在的应用开发和调试方式吗？例如本地开发、混合开发或者远程开发。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友一起阅读。我们下节课见。

分享给需要的人，Ta购买本课程，你将得 **18** 元

 生成海报并分享

 赞 2  提建议

上一篇

33 | 生搬硬套，如何搭建基于 J2EE 的分布式微服务

下一篇

35 | 秒级开发体验，如何实现容器热加载和一键调试？

更多课程推荐

李三红·搞定 Java 开发基础

极客时间 × 阿里云开发者社区联合出品

李三红
阿里云程序语言
与编译器技术总监
Java Champion



免费订阅 

精选留言 (2)

 写留言



bingo

2023-02-24 来自吉林

老师请问一下，使用Nocalhost远程debug Maven应用怎么在启动参数里设置自定义的setting.xml文件

作者回复: 在 nocalhost 的配置中 debug 的命令里写上业务启动命令，启动命令带上读取特定的配置文件。

实际上进入开发模式之后配置文件都有的，一般只要直接启动业务进程就行。



林龍

2023-02-24 来自广东

对于开发循环反馈这个流程，在微服务下是一定会遇到的问题，GitOps自动化是减少了整个的流程所花费的时间。在这三个虽然是解决方案，但是并不是最优秀的做法。其实对于该问题我觉得更应该从开发的角度去分析，

1.每个函数都有单元测试，确保每个函数的入参后得到的出参能符合预期

- 2.微服务的拆分，通过ddd的方式划分服务，尽可能把业务内的代码不要跨服务去处理。
- 3.在微服务中加入链路服务，可通过链路可视化工具把整个流程进行展示，然后对比各个微服务之间的入参出参是否与预期相同，如不同，则可对应到具体的微服务，具体的函数

当多个函数里面里面各个入参跟出参符合预期后，最终再把代码上传，构建，推送镜像，k8s启动。这样可以减少这个流程的次数。

作者回复: 很好的经验分享

在开发过程最主要还是要解决开发环境和编码查看代码效果的问题。

