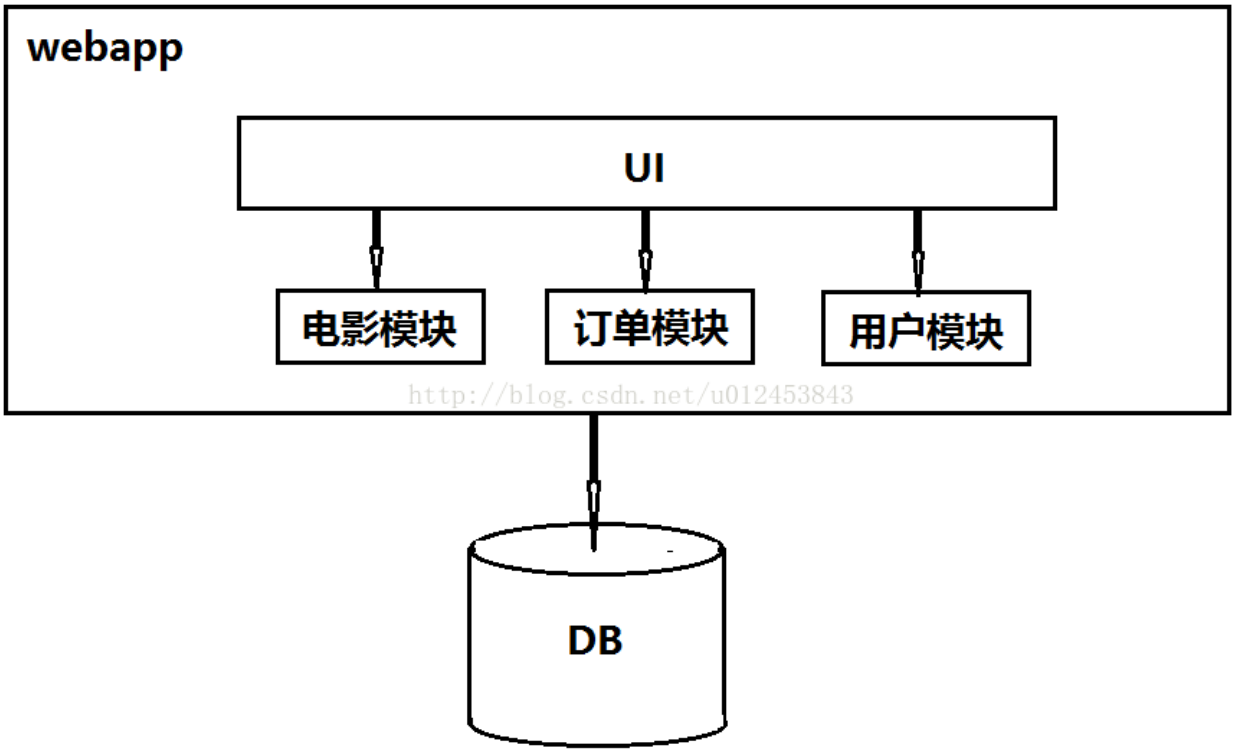
现在微服务这个名词越来越火了，公司最近也想使用微服务的技术，因此我就把我学习的东西记录下来，以备以后查询。既然要学习微服务，那么什么是微服务？微服务解决了什么问题？微服务有什么特点？就是我们首先要搞清楚的问题。

为了搞清楚什么是微服务我们首先来说一下传统的单体架构，一个归档包包含了应用所有功能的应用程序，我们通常称之为单体应用。架构单体应用的架构风格，我们称之为单体架构。如下图所示，一个Web工程包含了“电影模块”、“订单模块”、“用户模块”等多个模块，所有的模块都在同一个工程下，UI直接可以调用所有模块的接口，所有的模块共用一个数据库，这也是最常见的架构了。



单体架构在规模比较小的情况下工作情况良好，但是随着系统规模的扩大，它暴露出来的问题也越来越多，主要有以下几点：

1.复杂性逐渐变高

比如有的项目有几十万行代码，各个模块之间区别比较模糊，逻辑比较混乱，代码越多复杂性越高，越难解决遇到的问题。

2.技术债务逐渐上升

公司的人员流动是再正常不过的事情，有的员工在离职之前，疏于代码质量的自我管束，导致留下来很多坑，由于单体项目代码量庞大的惊人，留下的坑很难被发觉，这就给新来的员工带来很大的烦恼，人员流动越大所留下的坑越多，也就是所谓的技术债务越来越多。

3.部署速度逐渐变慢

这个就很好理解了，单体架构模块非常多，代码量非常庞大，导致部署项目所花费的时间越来越多，曾经有的项目启动就要一二十分钟，这是多么恐怖的事情啊，启动几次项目一天的时间就过去了，留给开发者开发的时间就非常少了。

4.阻碍技术创新

比如以前的某个项目使用struts2写的，由于各个模块之间有着千丝万缕的联系，代码量大，逻辑不够清楚，如果现在想用spring mvc来重构这个项目将是非常困难的，

付出的成本将非常大，所以更多的时候公司不得不硬着头皮继续使用老的struts架构，这就阻碍了技术的创新。

5.无法按需伸缩

比如说电影模块是CPU密集型的模块，而订单模块是IO密集型的模块，假如我们要提升订单模块的性能，比如加大内存、增加硬盘，但是由于所有的模块都在一个架构下，因此我们在扩展订单模块的性能时不得不考虑其它模块的因素，因为我们不能因为扩展某个模块的性能而损害其它模块的性能，从而无法按需进行伸缩。

既然单体架构存在着以上5点的缺陷，所以我们才需要改进架构，目前架构已从开始的单体架构演变到SOA直至今日的微服务架构。

说了上面那么多，那么到底什么是微服务？关于微服务目前业界没有一个统一的定义，引用Martin Fowler对微服务的描述：简而言之，微服务架构风格这种开发方法，是以开发一组小型服务的方式来开发一个独立的应用系统的。其中每个小型服务都运行在自己的进程中，并经常采用HTTP资源API这样轻量的机制来相互通信。这些服务围绕业务功能进行构建，并能通过全自动的部署机制来进行独立部署。这些微服务可以使用不同的语言来编写，并且可以使用不同的数据存储技术。对这些微服务我们仅做最低限度的集中管理。（来自：http://www.martinfowler.com/articles/microservices.html）

微服务具备的特性有哪些呢？

1.每个微服务可独立运行在自己的进程里；

2.一系列独立运行的微服务共同构建起了整个系统；

3.每个服务为独立的业务开发，一个微服务一般完成某个特定的功能，比如：订单管理，用户管理等；

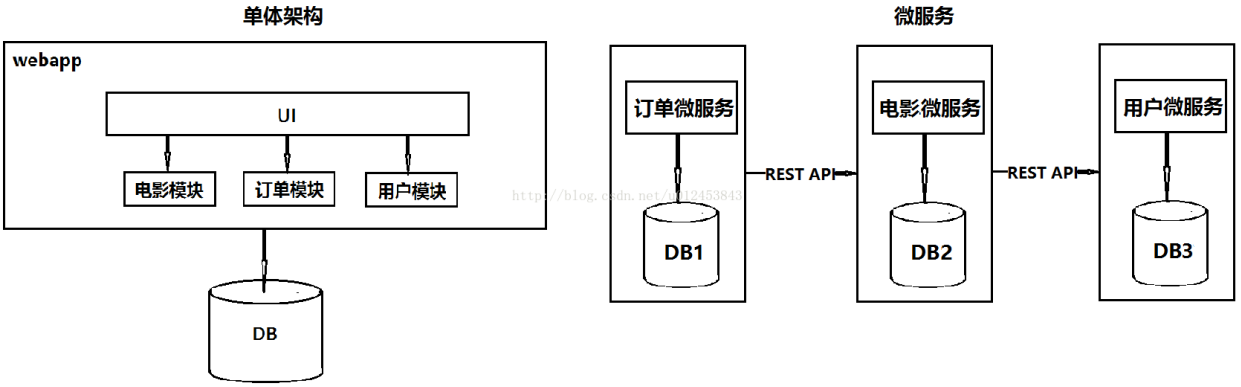
4.微服务之间通过一些轻量级的通信机制进行通信，例如通过REST API或者RPC的方式进行调用。

下面我们来看看单体架构和微服务的区别，如下图所示。

1.单体架构所有的模块全都耦合在一块，代码量大，维护困难，微服务每个模块就相当于一个单独的项目，代码量明显减少，遇到问题也相对来说比较好解决。

2.单体架构所有的模块都共用一个数据库，存储方式比较单一，微服务每个模块都可以使用不同的存储方式（比如有的用redis，有的用mysql等），数据库也是单个模块对应自己的数据库。

3.单体架构所有的模块开发所使用的技术一样，微服务每个模块都可以使用不同的开发技术，开发模式更灵活。



那么微服务的有点有哪些？

1.易于开发和维护

由于微服务单个模块就相当于一个项目，开发这个模块我们就只需关心这个模块的逻辑即可，代码量和逻辑复杂度都会降低，从而易于开发和维护。

2.启动较快

这是相对单个微服务来讲的，相比于启动单体架构的整个项目，启动某个模块的服务速度明显是要快很多的。

3.局部修改容易部署

在开发中发现了一个问题，如果是单体架构的话，我们就需要重新发布并启动整个项目，非常耗时间，但是微服务则不同，哪个模块出现了bug我们只需要解决那个模块的bug就可以了，解决完bug之后，我们只需要重启这个模块的服务即可，部署相对简单，不必重启整个项目从而大大节约时间。

4.技术栈不受限

比如订单微服务和电影微服务原来都是用java写的，现在我们想把电影微服务改成nodeJs技术，这是完全可以的，而且由于所关注的只是电影的逻辑而已，因此技术更换的成本也就会少很多。

5.按需伸缩

我们上面说了单体架构在想扩展某个模块的性能时不得不考虑到其它模块的性能会不会受影响，对于我们微服务来讲，完全不是问题，电影模块通过什么方式来提升性能不必考虑其它模块的情况。

6.DevOps

由于微服务架构下项目是靠多个单独的微服务共同运行的，那么部署微服务便成了问题，因为靠手工区部署的话，会浪费很多时间，这时自动化的部署方式便迫在眉睫了，DevOps便是帮我们自动部署微服务的，从而节约了我们的时间成本。

上面讲了那么多微服务的好处，我们知道，任何一个东西都不是完美的，有其优点便必然有其缺点，那么微服务的缺点有哪些呢？

1.运维要求较高

对于单体架构来讲，我们只需要维护好这一个项目就可以了，但是对于微服务架构来讲，由于项目是由多个微服务构成的，每个模块出现问题都会造成整个项目运行出现异常，想要知道是哪个模块造成的问题往往是不容易的，因为我们无法一步一步通过debug的方式来跟踪，这就对运维人员提出了很高的要求。

2.分布式的复杂性

对于单体架构来讲，我们可以不使用分布式，但是对于微服务架构来说，分布式几乎是必会用的技术，由于分布式本身的复杂性，导致微服务架构也变得复杂起来。

3.接口调整成本高

比如，用户微服务是要被订单微服务和电影微服务所调用的，一旦用户微服务的接口发生大的变动，那么所有依赖它的微服务都要做相应的调整，由于微服务可能非常多，那么调整接口所造成的成本将会明显提高。

4.重复劳动

对于单体架构来讲，如果某段业务被多个模块所共同使用，我们便可以抽象成一个工具类，被所有模块直接调用，但是微服务却无法这样做，因为这个微服务的工具类是

不能被其它微服务所直接调用的，从而我们便不得不在每个微服务上都建这么一个工具类，从而导致代码的重复。

微服务的设计原则：

1.单一职责原则

意思是每个微服务只需要实现自己的业务逻辑就可以了，比如订单管理模块，它只需要处理订单的业务逻辑就可以了，其它的不必考虑。

2.服务自治原则

意思是每个微服务从开发、测试、运维等都是独立的，包括存储的数据库也都是独立的，自己就有一套完整的流程，我们完全可以把它当成一个项目来对待。不必依赖于其它模块。

3.轻量级通信原则

首先是通信的语言非常的轻量，第二，该通信方式需要是跨语言、跨平台的，之所以要跨平台、跨语言就是为了让每个微服务都有足够的独立性，可以不受技术的钳制。

4.接口明确原则

由于微服务之间可能存在着调用关系，为了尽量避免以后由于某个微服务的接口变化而导致其它微服务都做调整，在设计之初就要考虑到所有情况，让接口尽量做的更通用，更灵活，从而尽量避免其它模块也做调整。

最后说一下微服务的开发框架，有以下四个。

1.Spring Cloud：http://projects.spring.io/spring-cloud（现在非常流行的微服务架构）

2.Dubbo：http：//dubbo.io

3.Dropwizard：http://www.dropwizard.io （关注单个微服务的开发）

4.Consul、etcd&etc.（微服务的模块）