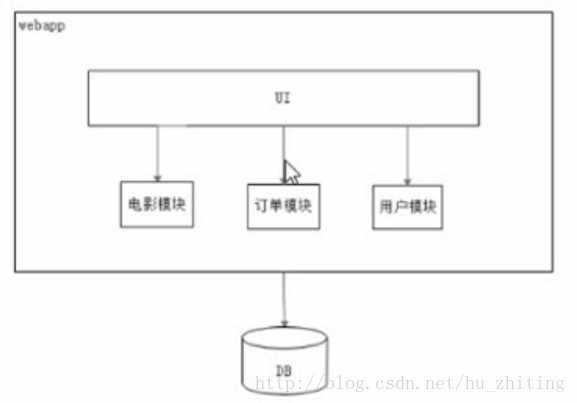
**单体架构与微服务架构**

**【单体架构】**

**1. 什么是单体架构**

**一个归档包（例如war格式）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。架构单体应用的方法论，我们称之为单体应用架构。**

**2. 单体架构示例图**



**3. 单体架构的缺陷**

**- 复杂性高。整个项目包含的模块非常多，模块的边界模糊，依赖关系不清晰，代码质量参差不齐……整个项目非常复杂。每次修改代码都心惊胆战，甚至添加一个简单的功能，或者修改一个BUG都会造成隐含的缺陷。**

**- 技术债务。随着时间推移、需求变更和人员更迭，会逐渐形成应用程序的技术债务，并且越积越多。已使用的系统设计或代码难以修改，因为应用程序的其他模块可能会以意料之外的方式使用它。**

**- 部署频率低。随着代码的增加，构建和部署的时间也会增加。而在单体应用中，每次功能的变更或缺陷的修复都会导致我们需要重新部署整个应用。全量部署的方式耗时长、影响范围大、风险高，这使得单体应用项目上线部署的频率较低，从而又导致两次发布之间会有大量功能变更和缺陷修复，出错概率较高。**

**- 扩展能力受限。单体应用只能作为一个整体进行扩展，无法结合业务模块的特点进行伸缩。**

**- 阻碍技术创新。单体应用往往使用统一的技术平台或方案解决所有问题，团队的每个成员都必须使用相同的开发语言和架构，想要引入新的框架或技术平台非常困难。**

**由于单体架构的缺陷日益明显，所以越来越多的公司采用微服务架构范式解决上面提到的单体架构中的问题。**

**不同于构建单一、庞大的应用，微服务架构将应用拆分为一套小且互相关联的服务。**

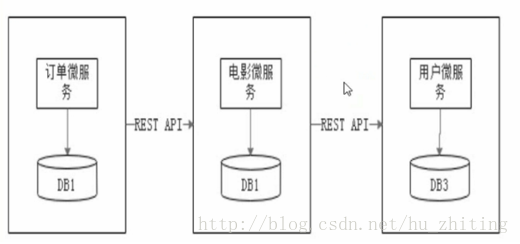
**【微服务架构】**

**1. 什么是微服务架构**

**简而言之，微服务架构风格的开发方法，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。**

**2. 微服务架构示例图**



**3. 微服务架构的特性**

**- 每个微服务可独立运行在自己的进程里；**

**- 一系列独立运行的微服务共同构建起整个系统；**

**- 每个服务为独立的业务开发，一个微服务只关注某个特定的功能，如订单管理、用户管理等；**

**- 微服务之间通过一些轻量的通信机制进行通信，如REST API进行调用；**

**- 可以使用不同的语言与存储技术；**

**- 全自动的部署机制；**

**4. 微服务架构的优势**

**- 易于开发和维护。一个微服务只关注一个特定的业务功能，所以它的业务清晰、代码量较少。开发和维护单个微服务相对比较简单，整个应用是由若干个微服务构建而成，所以整个应用也会维持在可控状态；**

**- 单个微服务启动较快。单个微服务代码量较少，所以启动会比较快；**

**- 局部修改容易部署。单体应用只要有修改，就要重新部署整个应用，微服务解决了这样的问题。一般来说，对某个微服务进行修改，只需要重新部署这个服务即可；**

**- 技术栈不受限。在微服务中，我们可以结合项目业务及团队的特点，合理地选择技术栈；**

**5. 微服务架构的挑战**

**- 运维要求较高。更多的服务意味着更多的运维投入。在单体架构中只需要保证一个应用的正常运行；而在微服务中，需要保证几十甚至几百个服务的正常运行与协作，带来了巨大的挑战；**

**- 分布式固有的复杂性。使用微服务构建的是分布式系统。对于一个分布式系统，系统容错、网络延迟、分布式事务等都带来了巨大的挑战；**

**- 接口调整成本高。微服务之间通过接口进行通信。如果修改某个微服务的API，可能所有使用了该接口的微服务都需要做调整；**

**- 重复劳动。很多服务可能都会使用到相同的功能。而这个功能并没有达到分解为一个微服务的程度，这个时候，可能各个服务都会开发这一功能，导致代码重复。**

**【总结】**

**构建复杂的应用的确非常困难。单体式的架构更适合轻量级的简单应用。如果你用它来开发复杂应用，那真的会很糟糕。微服务架构模式可以用来构建复杂应用，当然，这种架构模型也有自己的缺点和挑战。**