软件架构入门

作者： 阮一峰

日期： 2016年9月 3日

软件架构（software architecture）就是软件的基本结构。

合适的架构是软件成功的最重要因素之一。大型软件公司通常有专门的架构师职位（architect），只有资深程序员才可以担任。

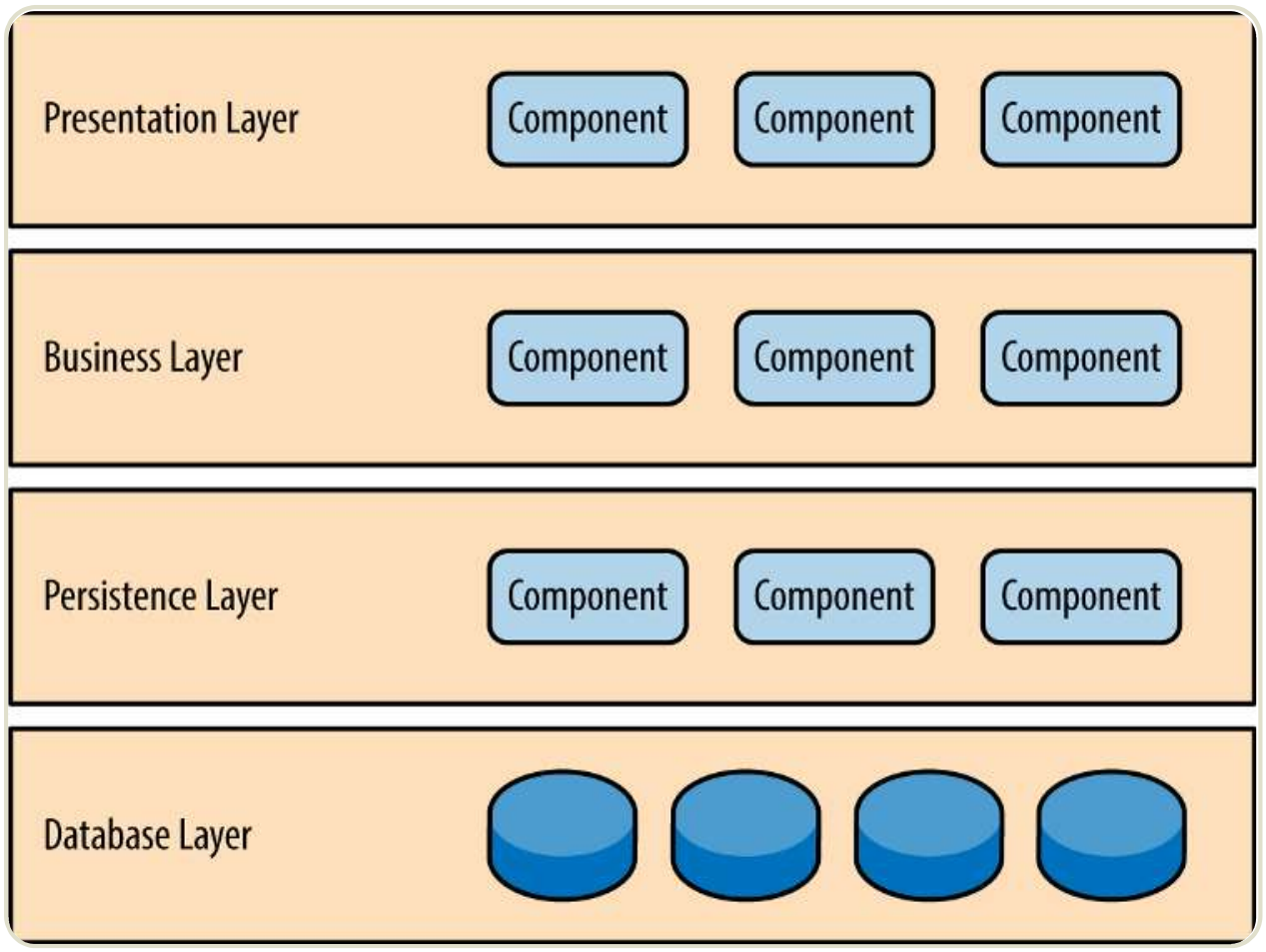
O'Reilly 出版过一本免费的小册子《Software Architecture Patterns》（PDF）， 介绍了五种最常见的软件架构，是非常好的入门读物。我读后受益匪浅，下面就是我的笔记。

一、分层架构

分层架构（layered architecture）是最常见的软件架构，也是事实上的标准架构。如果你不知道要用什么架构，那就用它。

这种架构将软件分成若干个水平层，每一层都有清晰的角色和分工，不需要知道其他层的细节。层与层之间通过接口通信。

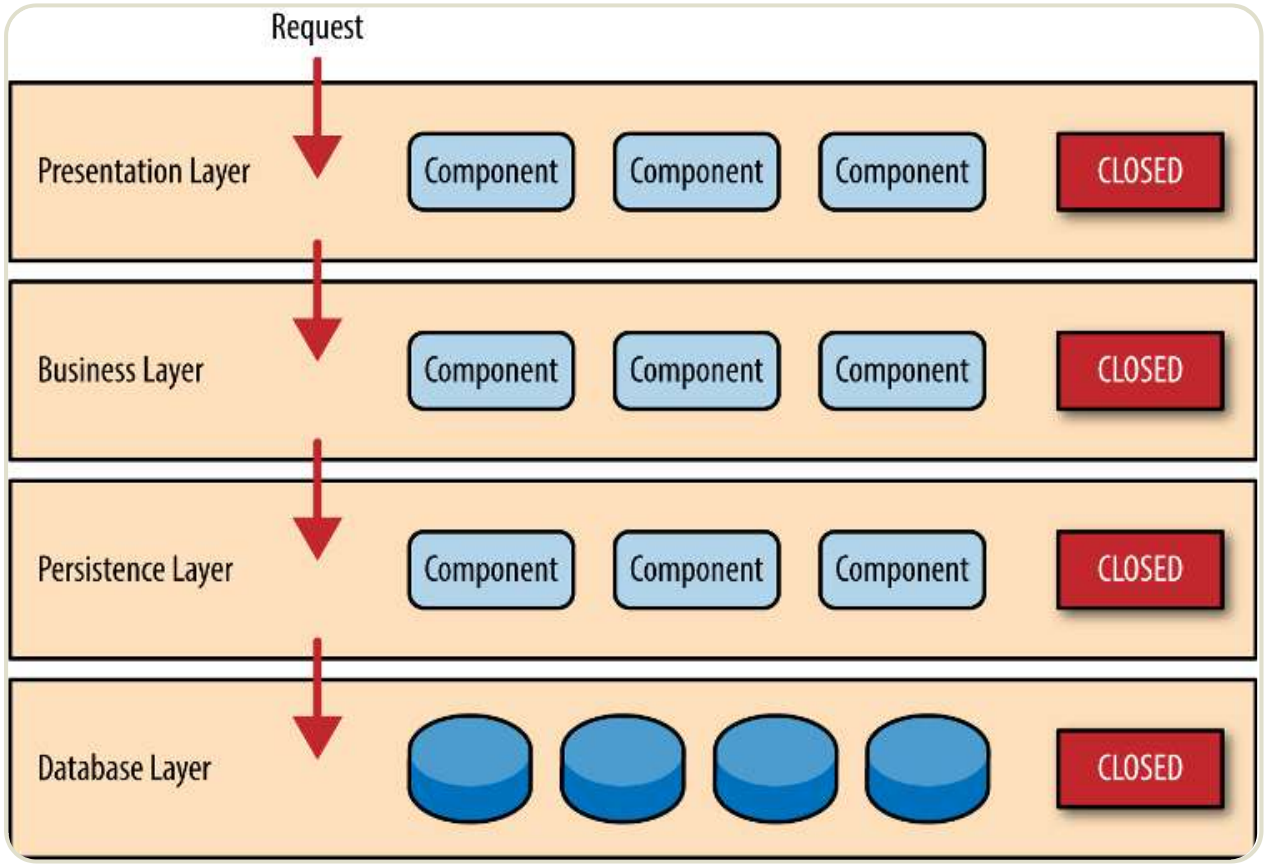
虽然没有明确约定，软件一定要分成多少层，但是四层的结构最常见。



- 表现层（presentation）：用户界面，负责视觉和用户互动
- 业务层（business）：实现业务逻辑
- 持久层（persistence）：提供数据，SQL 语句就放在这一层
- 数据库（database） ：保存数据

有的软件在逻辑层和持久层之间，加了一个服务层（service），提供不同业务逻辑需要的一些通用接口。

用户的请求将依次通过这四层的处理，不能跳过其中任何一层。



优点

- 结构简单，容易理解和开发
- 不同技能的程序员可以分工，负责不同的层，天然适合大多数软件公司的组织架构
- 每一层都可以独立测试，其他层的接口通过模拟解决

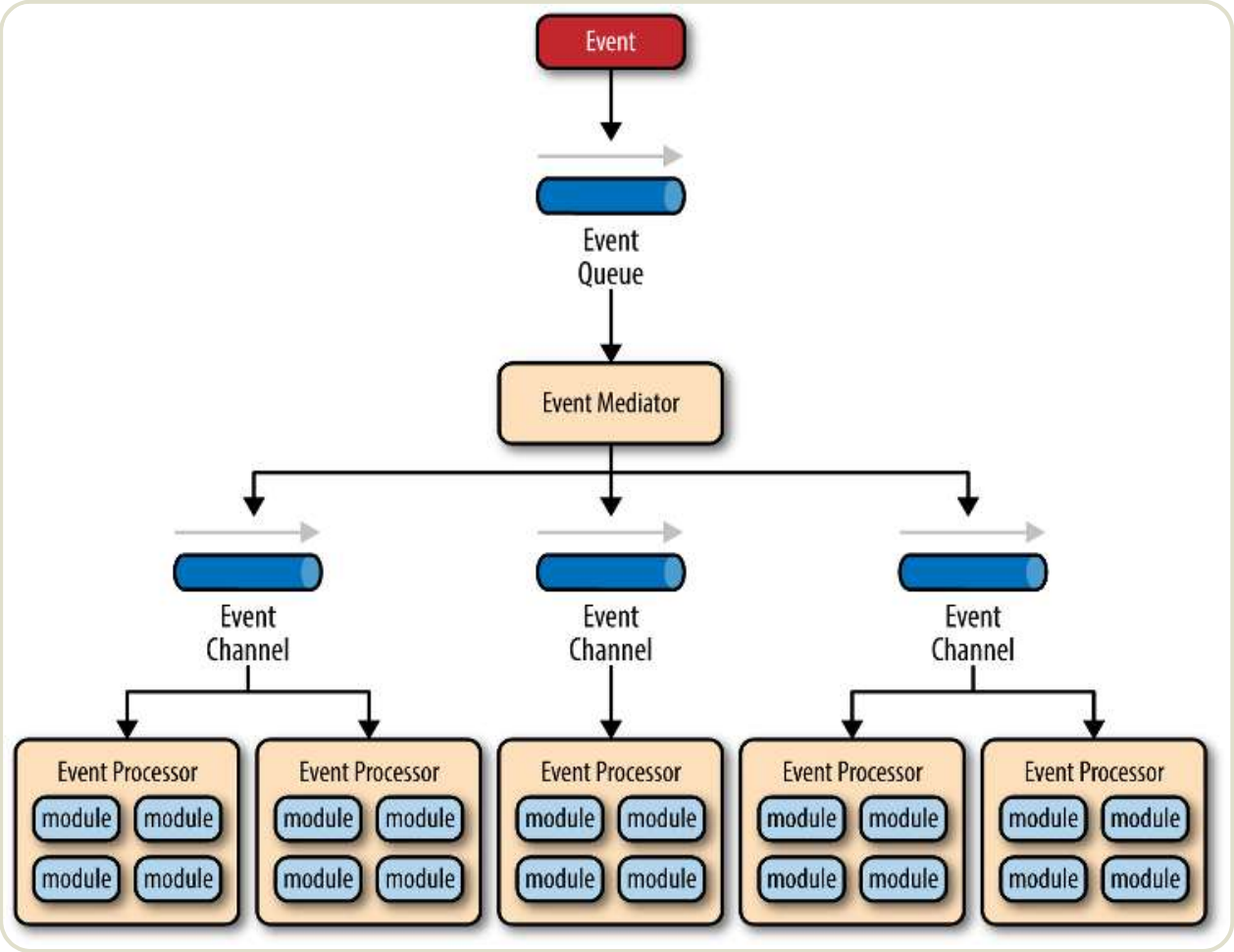
缺点

- 一旦环境变化，需要代码调整或增加功能时，通常比较麻烦和费时
- 部署比较麻烦，即使只修改一个小地方，往往需要整个软件重新部署，不容易做持续发布
- 软件升级时，可能需要整个服务暂停
- 扩展性差。用户请求大量增加时，必须依次扩展每一层，由于每一层内部是耦合的，扩展会很困难

二、事件驱动架构

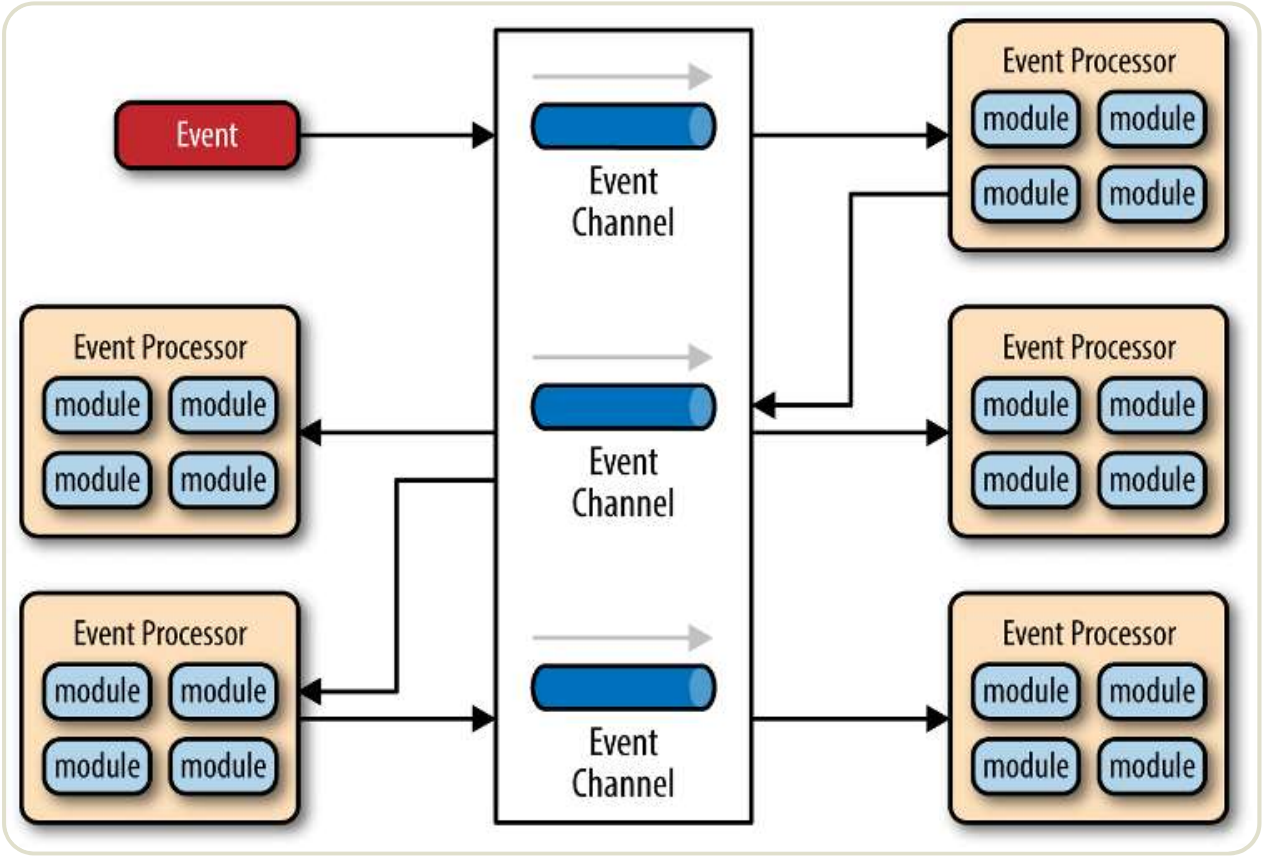
事件（event）是状态发生变化时，软件发出的通知。

事件驱动架构（event-driven architecture）就是通过事件进行通信的软件架构。它分成四个部分。



- 事件队列（event queue）：接收事件的入口
- 分发器（event mediator）：将不同的事件分发到不同的业务逻辑单元
- 事件通道（event channel）：分发器与处理器之间的联系渠道
- 事件处理器（event processor）：实现业务逻辑，处理完成后会发出事件，触发下一步操作

对于简单的项目，事件队列、分发器和事件通道，可以合为一体，整个软件就分成事件代理和事件处理器两部分。



优点

- 分布式的异步架构，事件处理器之间高度解耦，软件的扩展性好
- 适用性广，各种类型的项目都可以用
- 性能较好，因为事件的异步本质，软件不易产生堵塞
- 事件处理器可以独立地加载和卸载，容易部署

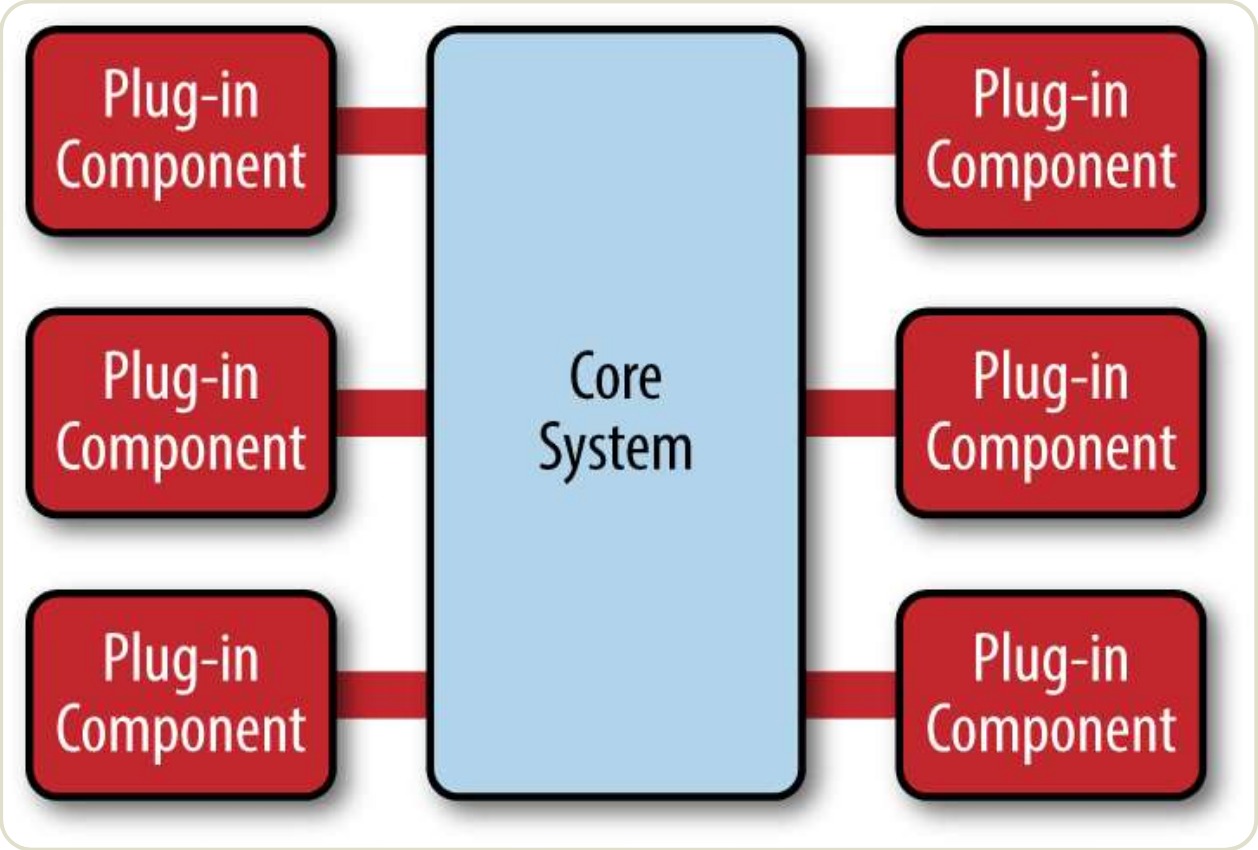
缺点

- 涉及异步编程（要考虑远程通信、失去响应等情况），开发相对复杂
- 难以支持原子性操作，因为事件通过会涉及多个处理器，很难回滚
- 分布式和异步特性导致这个架构较难测试

三、微核架构

微核架构（microkernel architecture）又称为"插件架构"（plug-in architecture），指的是软件的内核相对较小，主要功能和业务逻辑都通过插件实现。

内核（core）通常只包含系统运行的最小功能。插件则是互相独立的，插件之间的通信，应该减少到最低，避免出现互相依赖的问题。



优点

- 良好的功能延伸性（`extensibility`），需要什么功能，开发一个插件即可
- 功能之间是隔离的，插件可以独立的加载和卸载，使得它比较容易部署，
- 可定制性高，适应不同的开发需要
- 可以渐进式地开发，逐步增加功能

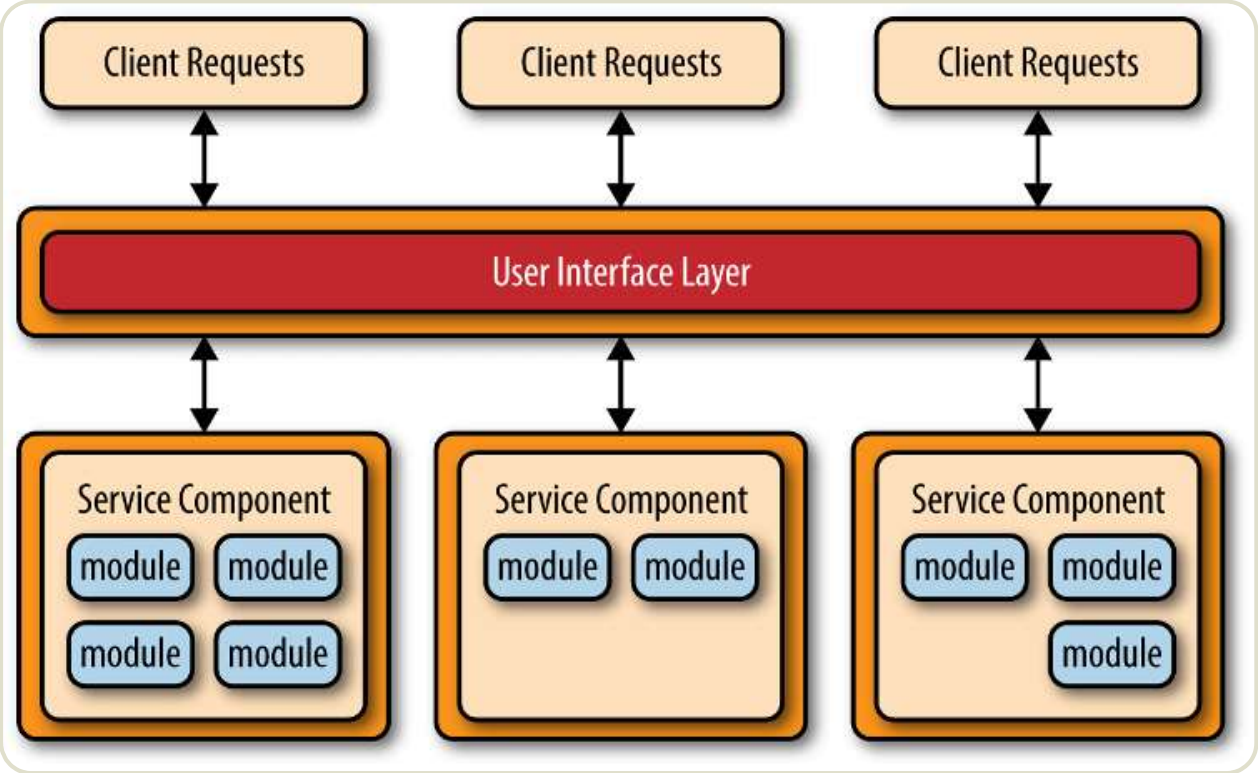
缺点

- 扩展性（`scalability`）差，内核通常是一个独立单元，不容易做成分布式
- 开发难度相对较高，因为涉及到插件与内核的通信，以及内部的插件登记机制

四、微服务架构

微服务架构（microservices architecture）是服务导向架构（service-oriented architecture，缩写 SOA）的升级。

每一个服务就是一个独立的部署单元（separately deployed unit）。这些单元都是分布式的，互相解耦，通过远程通信协议（比如REST、SOAP）联系。



微服务架构分成三种实现模式。

- RESTful API 模式：服务通过 API 提供，云服务就属于这一类
- RESTful 应用模式：服务通过传统的网络协议或者应用协议提供，背后通常是一个多功能的应用程序，常见于企业内部
- 集中消息模式：采用消息代理（message broker），可以实现消息队列、负载均衡、统一日志和异常处理，缺点是会出现单点失败，消息代理可能要做成集群

优点

- 扩展性好，各个服务之间低耦合
- 容易部署，软件从单一可部署单元，被拆成了多个服务，每个服务都是可部署单元
- 容易开发，每个组件都可以进行持续集成式的开发，可以做到实时部署，不间断地升级
- 易于测试，可以单独测试每一个服务

缺点

- 由于强调互相独立和低耦合，服务可能会拆分得很细。这导致系统依赖大量的微服务，变得很凌乱和笨重，性能也会不佳。
- 一旦服务之间需要通信（即一个服务要用到另一个服务），整个架构就会变得复杂。典型的例子就是一些通用的 Utility 类，一种解决方案是把它们拷贝到每一个服务中

去，用冗余换取架构的简单性。

- 分布式的本质使得这种架构很难实现原子性操作，交易回滚会比较困难。

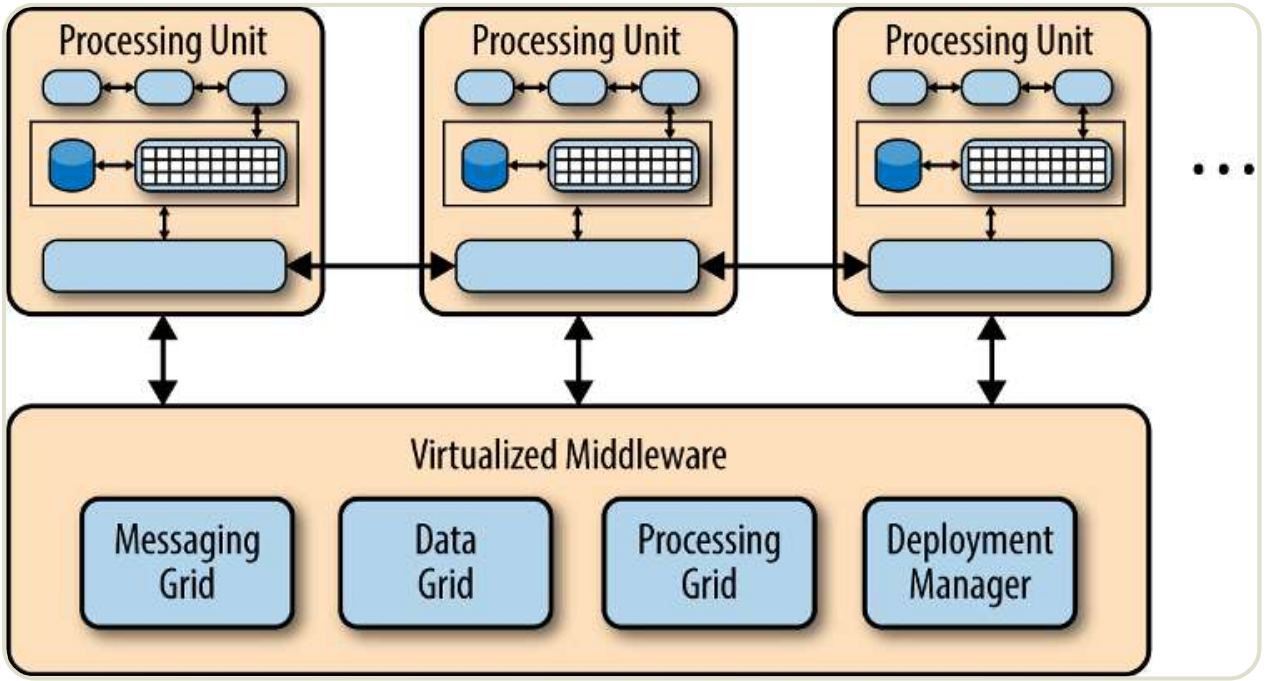
五、云架构

云结构（cloud architecture）主要解决扩展性和并发的问题，是最容易扩展的架构。

它的高扩展性，主要原因是没使用中央数据库，而是把数据都复制到内存中，变成可复制的内存数据单元。然后，业务处理能力封装成一个个处理单元（prcessing unit）。访问量增加，就新建处理单元；访问量减少，就关闭处理单元。由于没有中央数据库，所以扩展性的最大瓶颈消失了。由于每个处理单元的数据都在内存里，最好要进行数据持久化。

这个模式主要分成两部分：处理单元（processing unit）和虚拟中间件（virtualized middleware）。

- 处理单元：实现业务逻辑
- 虚拟中间件：负责通信、保持sessions、数据复制、分布式处理、处理单元的部署。



虚拟中间件又包含四个组件。

- 消息中间件（Messaging Grid）：管理用户请求和session，当一个请求进来以后，决定分配给哪一个处理单元。

- 数据中间件（Data Grid）：将数据复制到每一个处理单元，即数据同步。保证某个处理单元都得到同样的数据。
- 处理中间件（Processing Grid）：可选，如果一个请求涉及不同类型的处理单元，该中间件负责协调处理单元
- 部署中间件（Deployment Manager）：负责处理单元的启动和关闭，监控负载和响应时间，当负载增加，就新启动处理单元，负载减少，就关闭处理单元。

优点

- 高负载，高扩展性
- 动态部署

缺点

- 实现复杂，成本较高
- 主要适合网站类应用，不合适大量数据吞吐的大型数据库应用
- 较难测试

（完）

文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名（创意共享3.0许可证）
- 发表日期： 2016年9月 3日

相关文章

- **2022.06.29:** 云主机上手教程：轻量应用服务器体验

  > 很多同学都希望架设自己的云服务，这就离不开云主机（cloud server）。

- **2022.04.29:** 微服务是什么？

  > 微服务（microservice）是一种软件架构，正得到越来越多的关注。

- **2022.01.28:** 命令行常用工具的替代品

> 程序员离不开命令行，许多经典命令是每天必用的，比如ls和cd。

- **2021.09.07:** 《C 语言入门教程》发布了

> 向大家报告，我写了一本《C 语言入门教程》，已经上线了，欢迎访问。

Weibo | Twitter | GitHub

Email: yifeng.ruan@gmail.com