**微服务数据库**

每个微服务都有自己独立的数据库，服务A想访问服务B的数据必须通过B的Api进行访问

**逻辑架构和物理架构并不是一一对应**

如服务A并不一定部署在Docker中

**分布式数据管理的挑战**

1. 如何微服务边界
2. 如何创建从多个微服务中获取数据的查询
3. 如何在多个微服务之间实现一致性
4. 多个微服务之间的通信

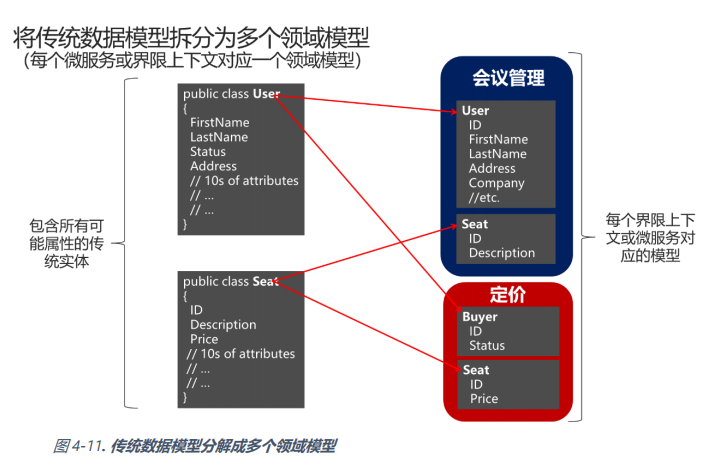
**识别微服务的领域模型边界**

微服务依据界限上下文进行划分（界限上下文划分请搜索上下文映射模式）

**数据模型拆分成多个领域模型**

如下，将User和Seat 2个实体拆分到2个微服务中，每个微服务的实体只关心自己所需的字段，微服务的模型里无需包含任何不需要的数据。

如，用户在“会议管理”微服务里表示为“User”实体，在“定价微服务”里以“Buyer”实体来体现。

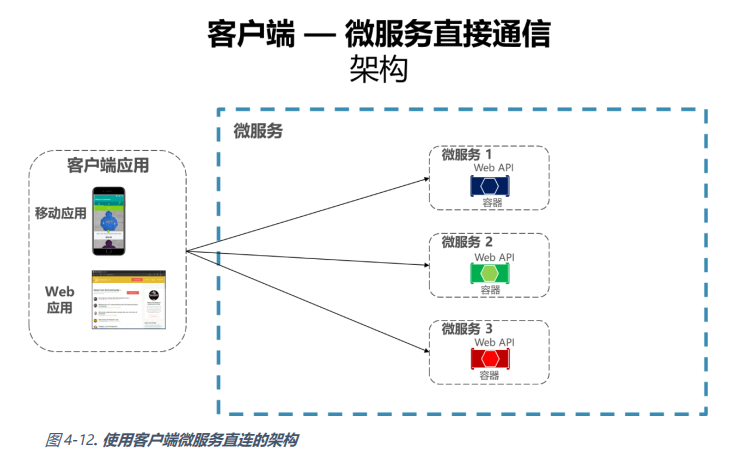


**客户端与微服务通信**

客户端如何与微服务通信，下面给出了2中模式

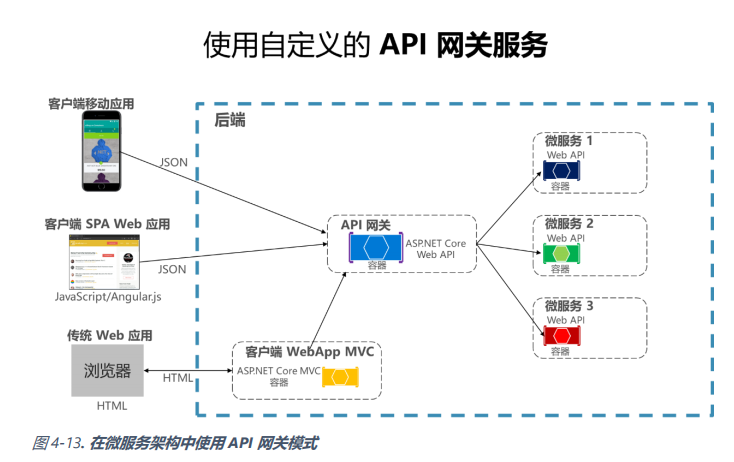
1. 客户端微服务直连

客户端直接调用微服务



1. 使用 API 网关

当设计和创建支持多种客户端的大型或复杂微服务应用时，一种好办法是使用 API 网关



通常来说，使用一个 API 网关将应用中所有内部微服务聚合在一起，这种做法并不明智。如果这么做，就等同于创建了一个将所有微服务耦合在一起的单体聚合器或编排引擎，这违反了微服务的自治原则。因此，API 网关应该基于业务边界来拆分，而不是作为整个应用的一个聚合器。

我们应该使用多个个Api网关来映射到我们不同的服务

**微服务间的通信**

1. 同步与异步通信

同步通信是指服务A必须等待服务B返回结果才会往下执行

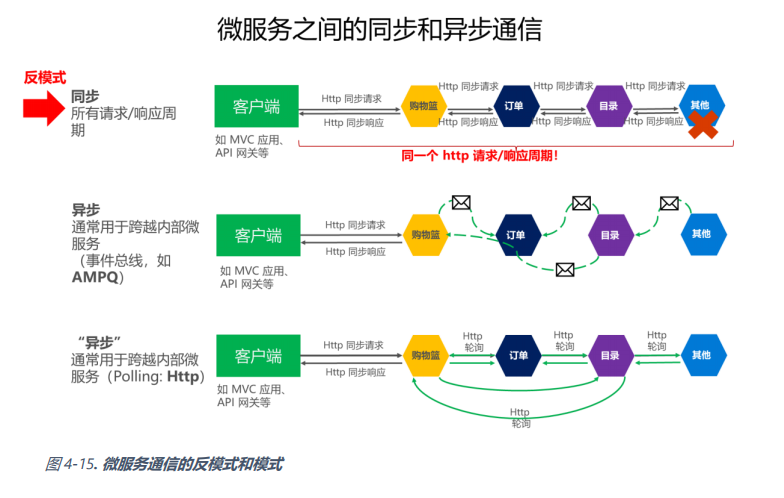
异步通信是指服务A不需要等待服务B返回结果

1. 微服务应该避免同步通信

微服务是一个独立的个体，即使服务B不存在，服务A也可以正常运行

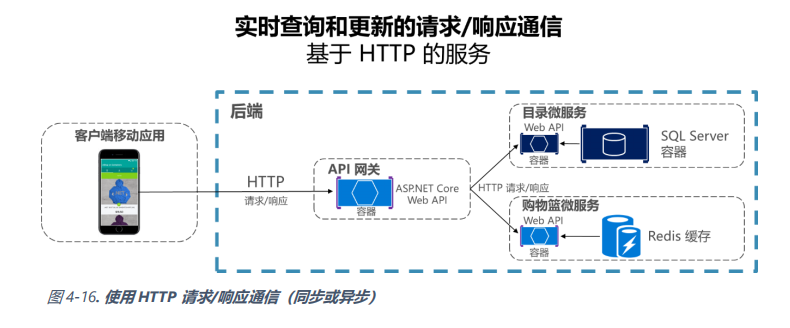
1. 异步通信推荐使用消息队列（如RabbitMQ）

服务A发生某个事件时发生消息到消息队列，服务B接受到消息就会执行，服务B完成后发生完成消息到消息队列，服务A收到后会执行接下来的操作



**使用 HTTP 和 REST 的请求/响应通信**

请求/响应通信尤其适合客户端应用的实时数据查询 UI（实时界面）。因此在微服务架构中，我们可能会使用这种通信机制来处理大部分查询。如图 4-16 所示。

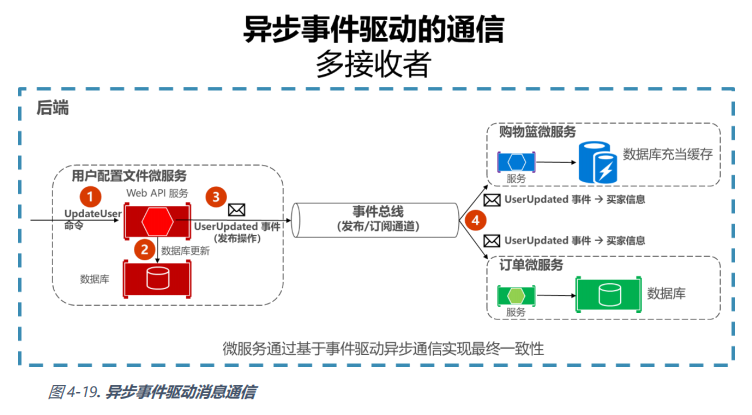


当客户端使用请求/响应通信时，前提认为响应内容会在很短时间内返回，通常低于 1 秒，最多几秒时 间。

**异步消息通信**

跨越多个微服务及其相关领域模型传送变化时，使用异步消息和事件驱动的通信至关重要，如使用RabbitMQ。

使用异步通信我们尽可能遵循一个规则：只在内部服务间使用异步消息传递，只在从客户端应用到前端服务（API 网关加上第一级微服务）间使用同步通信（如 HTTP）



使用异步通信，当微服务A的领域实体发生改变时，可能会发生某个事件到事件总线上，微服务A并不关系谁会处理该事件，微服务B如果订阅了该事件，那么他会处理该事件，如果处理过程中，微服务B触发了某个领域事件，他同样会发生事件到事件总线上，其他服务会处理