**高并法系统的设计**

# 基础篇

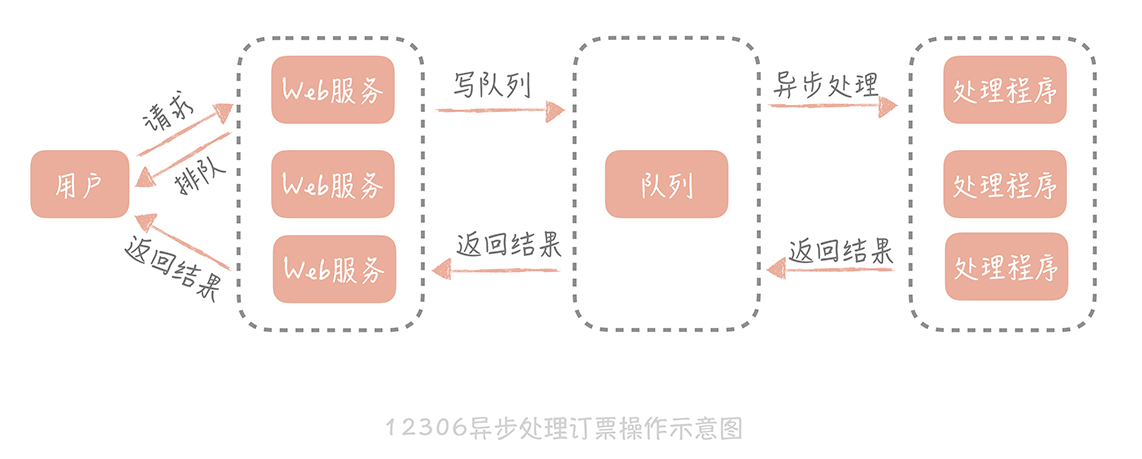
**Scale-up： 在我们的系统设计之初会考虑Scale,因为这个方案简单，能够用堆砌硬件的方案解决问题，就用硬件来解决。**

**Scale-out ： 当系统的并发达到单机的物理极限的时，我们就要使用Scale-out。**

**缓存：普通磁盘在扇区上的的寻址时间是10ms，内存在微秒级别，cpu在纳妙级别**

**千兆网卡上的读取时间也是微妙级别。**

**异步处理：**

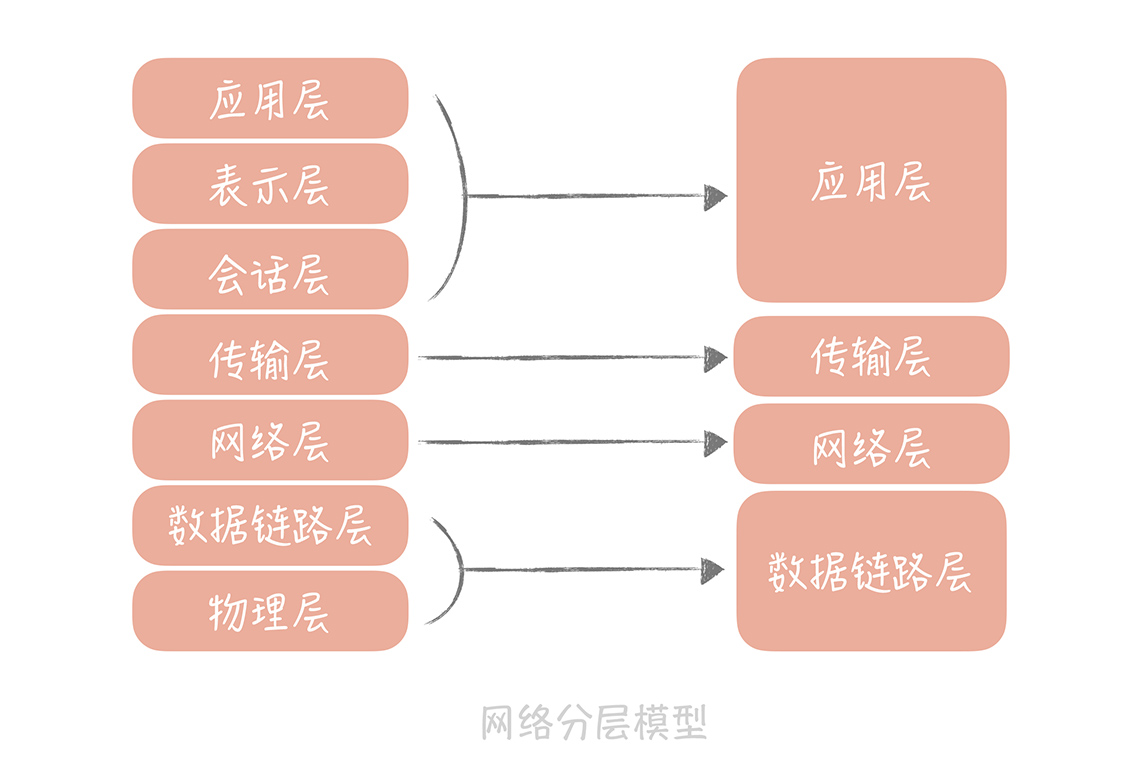


**一般系统的演进过程应该遵循的如下思路：**

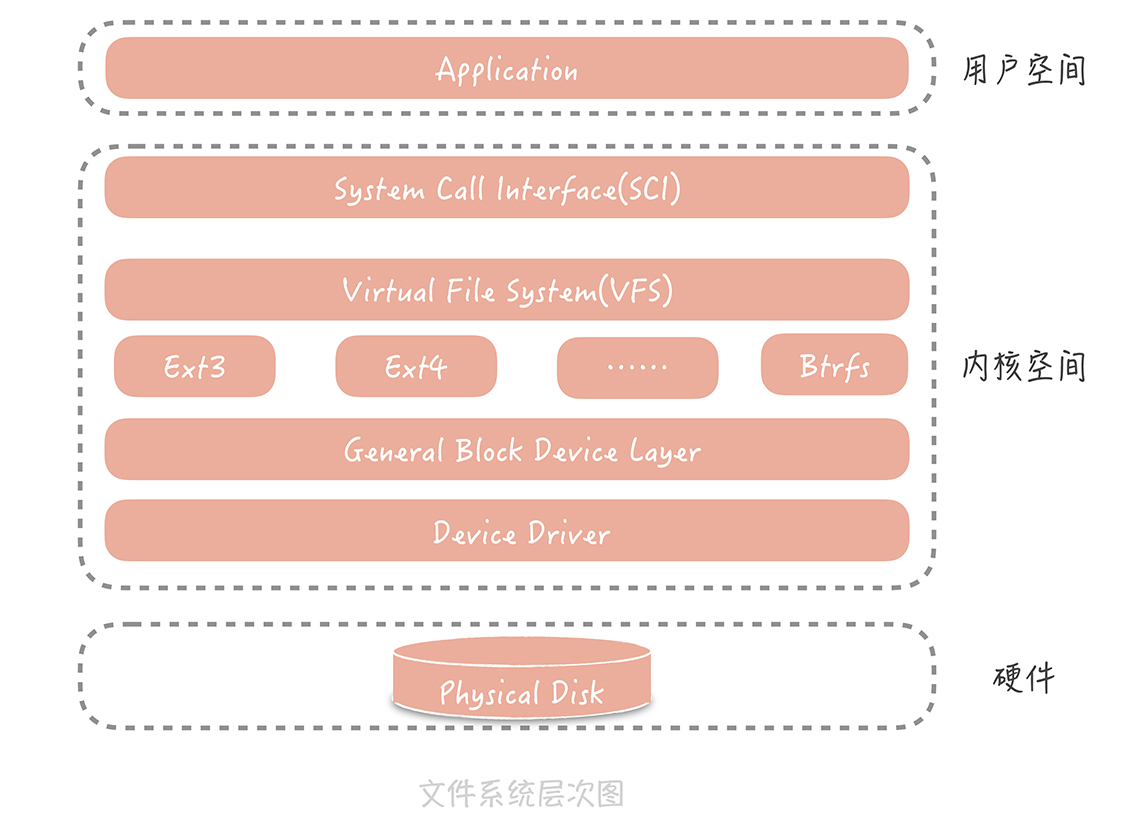
1. **最简单的系统设计满足业务需求和流量现状。**
2. **随着流量的增加和业务的变化修正架构中存在问题的点，如单点问题，横向扩展问题，性能无法满足需求的组件，在这个过程中，选择社区成熟的，团队熟悉的组件帮我们解决问题，在社区没有合适的解决方案前提下才会自己造轮子。**
3. **当对架构的小修小补无法满足需求时候，考虑重构、重写等大的调整方式解决现有的问题。**

**高并法系统的演进应该是循序渐进的，以解决系统存在的问题为目的和驱动力的。**

**OSI网路模型：**



**Linux 系统分层结构：**



**分层的好处：**

**分层可以简化系统设计，让不同的人专注做某一层次的事。**

**分层之后可以做到和你高的复用性。**

**分层架构可以让我们更容易的做横向扩展。**

**如果来做系统分层：**

**高并法系统设计的三大目标：**

**高性能、高可用、高扩展**

**高并法：是指用设计手段让系统能够处理更多的用户请求，也就是承担更大的用户流量，他是一切架构设计的背景和前提，性能和可用性，是我们实现高并发系统设计必须考虑的因素。**

**高性能，高可用，可扩展是我们设计高并发系统时追求的的三个目标。**

**性能优化原则：**

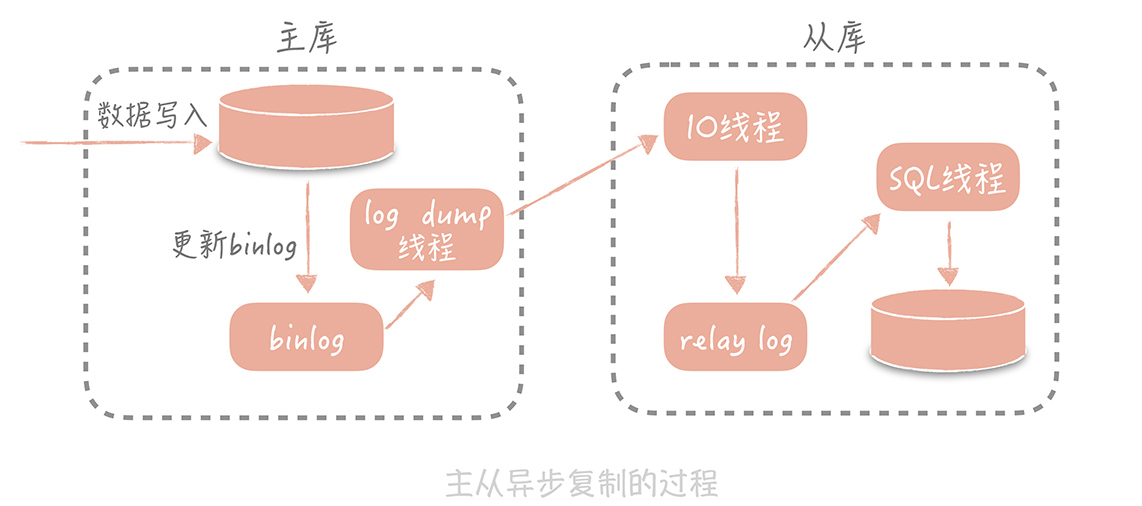
1. **性能优化一定是问题为导向的，不能盲目。**
2. **性能优化遵循“二八原则”用20%的精力去解决80%的性能问题。**
3. **性能的优化要有数据的支撑。**
4. **性能优化的过程是持续的。**
5. **性能的度量指标是什么。**

**高并下的性能优化：**

**主从读写分离的两个关键技术点：**

1. 数据的拷贝，我们称之为主从复制。
2. 在主从分离的情况下，我们如何屏蔽主从分离带来的访问数据库方式的变化，让开发人员感觉在使用单一数据库一样。

（1）主从复制：**主从复制的过程是这样的**：首先从库在连接到主节点时会创建一个 IO 线程，用以请求主库更新的 binlog，并且把接收到的 binlog 信息写入一个叫做 relay log 的日志文件中，而主库也会创建一个 log dump 线程来发送 binlog 给从库；同时，从库还会创建一个 SQL 线程读取 relay log 中的内容，并且在从库中做回放，最终实现主从的一致性。这是一种比较常见的主从复制方式。



**主从同步的延迟，是我们排查问题时很容易忽略的一个问题：**

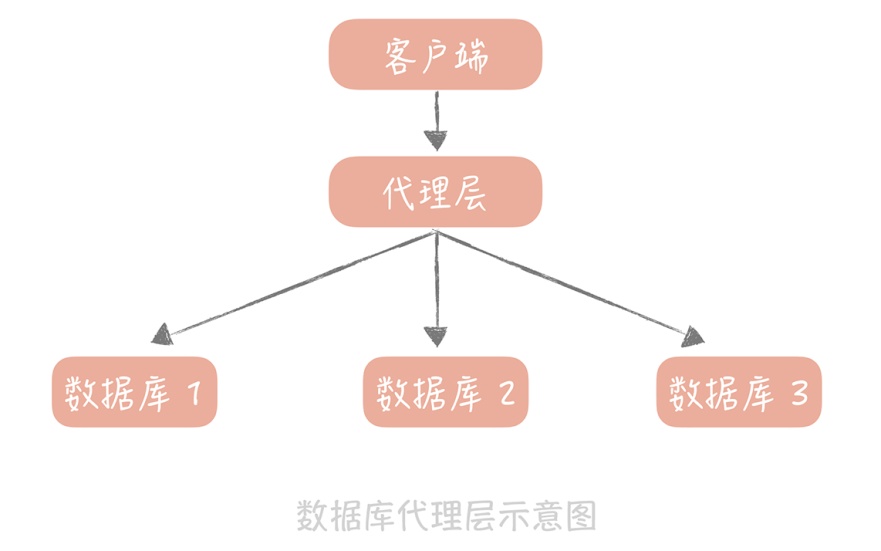
**如何访问数据库：**

为了降低实现的复杂度，业界涌现了很多数据库中间件来解决数据库的访问问题，这些中间件可以分为两类。

第一类以淘宝的 TDDL（ Taobao Distributed Data Layer）为代表。

另一类：是单独部署的代理层方案，这一类方案代表比较多，如早期阿里巴巴开源的 Cobar，基于 Cobar 开发出来的 Mycat，360 开源的 Atlas，美团开源的基于 Atlas 开发的 DBProxy 等等。

这一类中间件部署在独立的服务器上，业务代码如同在使用单一数据库一样使用它，实际上它内部管理着很多的数据源，当有数据库请求时，它会对 SQL 语句做必要的改写，然后发往指定的数据源。



**1. 主从读写分离以及部署一主多从可以解决突发的数据库读流量，是一种数据库横向扩展的方法；**

**2. 读写分离后，主从的延迟是一个关键的监控指标，可能会造成写入数据之后立刻读的时候读取不到的情况；**

**3. 业界有很多的方案可以屏蔽主从分离之后数据库访问的细节，让开发人员像是访问单一数据库一样，包括有像 TDDL、Sharding-JDBC 这样的嵌入应用内部的方案，也有像 Mycat 这样的独立部署的代理方案。**

