二

13 架构设计流程：详细方案设计

13 架构设计流程：详细方案设计完成备选方案的设计和选择后，我们终于可以长出一口气，因为整个架构设计最难的一步已经完成了，但整体方案尚未完成，架构师还需继续努力。接下来我们需要再接再励，将最终确定的备选方案进行细化，使得备选方案变成一个可以落地的设计方案。所以今天我来讲讲架构设计流程第 4 步：详细方案设计。

架构设计第 4 步：详细方案设计

简单来说，详细方案设计就是将方案涉及的关键技术细节给确定下来。

假如我们确定使用 Elasticsearch 来做全文搜索，那么就需要确定 Elasticsearch 的索引是按照业务划分，还是一个大索引就可以了；副本数量是 2 个、3 个还是 4 个，集群节点数量是 3 个还是 6 个等。

假如我们确定使用 MySQL 分库分表，那么就需要确定哪些表要分库分表，按照什么维度来分库分表，分库分表后联合查询怎么处理等。

假如我们确定引入 Nginx 来做负载均衡，那么 Nginx 的主备怎么做，Nginx 的负载均衡策略用哪个（权重分配？轮询？ip_hash？）等。

可以看到，详细设计方案里面其实也有一些技术点和备选方案类似。例如，Nginx 的负载均衡策略，备选有轮询、权重分配、ip_hash、fair、url_hash 五个，具体选哪个呢？看起来和备选方案阶段面临的问题类似，但实际上这里的技术方案选择是**很轻量级的**，我们无须像备选方案阶段那样操作，而只需要简单根据这些技术的适用场景选择就可以了。

例如，Nginx 的负载均衡策略，简单按照下面的规则选择就可以了。

每个请求按时间顺序逐一分配到不同的后端服务器，后端服务器分配的请求数基本一致，如果后端服务器“down 掉”，能自动剔除。

根据权重来进行轮询，权重高的服务器分配的请求更多，主要适应于后端服务器性能不均的情况，如新老服务器混用。

每个请求按访问 IP 的 hash 结果分配，这样每个访客固定访问一个后端服务器，主要用于解决 session 的问题，如购物车类的应用。

按后端服务器的响应时间来分配请求，响应时间短的优先分配，能够最大化地平衡各后端服务器的压力，可以适用于后端服务器性能不均衡的情况，也可以防止某台后端服务器性能不足的情况下还继续接收同样多的请求从而造成雪崩效应。

按访问 URL 的 hash 结果来分配请求，每个 URL 定向到同一个后端服务器，适用于后端服务器能够将 URL 的响应结果缓存的情况。

这几个策略的适用场景区别还是比较明显的，根据我们的业务需要，挑选一个合适的即可。例如，比如一个电商架构，由于和 session 比较强相关，因此如果用 Nginx 来做集群负载均衡，那么选择 ip_hash 策略是比较合适的。

详细设计方案阶段可能遇到的一种极端情况就是在详细设计阶段发现备选方案不可行，一般情况下主要的原因是备选方案设计时遗漏了某个关键技术点或者关键的质量属性。例如，我曾经参与过一个项目，在备选方案阶段确定是可行的，但在详细方案设计阶段，发现由于细节点太多，方案非常庞大，整个项目可能要开发长达 1 年时间，最后只得废弃原来的备选方案，重新调整项目目标、计划和方案。这个项目的主要失误就是在备选方案评估时忽略了开发周期这个质量属性。

幸运的是，这种情况可以通过下面方式有效地避免：

架构师不但要进行备选方案设计和选型，还需要对备选方案的关键细节有较深入的理解。例如，架构师选择了 Elasticsearch 作为全文搜索解决方案，前提必须是架构师自己对 Elasticsearch 的设计原理有深入的理解，比如索引、副本、集群等技术点；而不能道听途说 Elasticsearch 很牛，所以选择它，更不能成为把“细节我们不讨论”这句话挂在嘴边的“PPT 架构师”。

通过分步骤、分阶段、分系统等方式，尽量降低方案复杂度，方案本身的复杂度越高，某个细节推翻整个方案的可能性就越高，适当降低复杂性，可以减少这种风险。

如果方案本身就很复杂，那就采取**设计团队**的方式来进行设计，博采众长，汇集大家的智慧和经验，防止只有 1~2 个架构师可能出现的思维盲点或者经验盲区。

详细方案设计实战

虽然我们上期在“前浪微博”消息队列的架构设计挑选了备选方案 2 作为最终方案，但备选方案设计阶段的方案粒度还比较粗，无法真正指导开发人员进行后续的设计和开发，因此需要在备选方案的基础上进一步细化。

下面我列出一些备选方案 2 典型的需要细化的点供参考，有兴趣的同学可以自己尝试细化更多的设计点。

1. 细化设计点 1：数据库表如何设计？

数据库设计两类表，一类是日志表，用于消息写入时快速存储到 MySQL 中；另一类是消息表，每个消息队列一张表。

业务系统发布消息时，首先写入到日志表，日志表写入成功就代表消息写入成功；后台线程再从日志表中读取消息写入记录，将消息内容写入到消息表中。

业务系统读取消息时，从消息表中读取。

日志表表名为 MQ_LOG，包含的字段：日志 ID、发布者信息、发布时间、队列名称、消息内容。

消息表表名就是队列名称，包含的字段：消息 ID（递增生成）、消息内容、消息发布时间、消息发布者。

日志表需要及时清除已经写入消息表的日志数据，消息表最多保存 30 天的消息数据。

2. 细化设计点 2：数据如何复制？

直接采用 MySQL 主从复制即可，只复制消息存储表，不复制日志表。

3. 细化设计点 3：主备服务器如何倒换？

采用 ZooKeeper 来做主备决策，主备服务器都连接到 ZooKeeper 建立自己的节点，主服务器的路径规则为“/MQ/server/ 分区编号 /master”，备机为“/MQ/server/ 分区编号 /slave”，节点类型为 EPHEMERAL。

备机监听主机的节点消息，当发现主服务器节点断连后，备服务器修改自己的状态，对外提供消息读取服务。

4. 细化设计点 4：业务服务器如何写入消息？

消息队列系统设计两个角色：生产者和消费者，每个角色都有唯一的名称。

消息队列系统提供 SDK 供各业务系统调用，SDK 从配置中读取所有消息队列系统的服务器信息，SDK 采取轮询算法发起消息写入请求给主服务器。如果某个主服务器无响应或者返回错误，SDK 将发起请求发送到下一台服务器。

5. 细化设计点 5：业务服务器如何读取消息？

消息队列系统提供 SDK 供各业务系统调用，SDK 从配置中读取所有消息队列系统的服务器信息，轮流向所有服务器发起消息读取请求。

消息队列服务器需要记录每个消费者的消费状态，即当前消费者已经读取到了哪条消息，当收到消息读取请求时，返回下一条未被读取的消息给消费者。

6. 细化设计点 6：业务服务器和消息队列服务器之间的通信协议如何设计？

考虑到消息队列系统后续可能会对接多种不同编程语言编写的系统，为了提升兼容性，传输协议用 TCP，数据格式为 ProtocolBuffer。

当然还有更多设计细节就不再一一列举，因此这还不是一个完整的设计方案，我希望可以通过这些具体实例来说明细化方案具体如何去做。

小结

今天我为你讲了架构设计流程的第四个步骤：详细方案设计，并且基于模拟的“前浪微博”消息队列系统，给出了具体的详细设计示例，希望对你有所帮助。这个示例并不完整，有兴趣的同学可以自己再详细思考一下还有哪些细节可以继续完善。

这就是今天的全部内容，留一道思考题给你吧，你见过“PPT 架构师”么？他们一般都具备什么特点？

上一页 下一页