23 | 重构：烟囱式、平台化、中台化的架构同与异

上一讲里，我们介绍了两大类型的系统升级重构方案，还介绍了如何进行重构版本的上线，以及如何平滑地完成新老版本切换的方案。在本讲里，将会具体介绍如何判断系统发展到什么阶段需要重构，以及如何实施重构。

系统稳定性的重构升级

简单、通用的微服务架构如下图 1 所示，它包含一个应用服务和一个数据库作为存储。

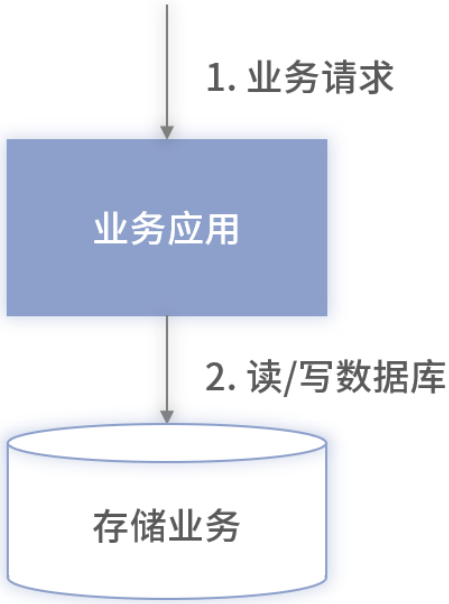

图 1：简单、通用的微服务架构图

当上图中展示的架构出现如下一些问题时，可以采用上一讲中提到的“微服务中纯代码维度的升级重构”。

1. 代码日志打印冗余，且因为直接使用大对象进行 JSON 序列化的方式打印日志，进而导致常常出现 CPU 飙升。
2. 代码中未增加监控报警，导致用户先感知线上问题，研发再进行修复。
3. 代码可维护性低，开发需求耗时长，且开发时代码“牵一发而动全身”，产生的 Bug 较多。具体的一些表现是：

（1）一个类中有上千或者上万行代码；

（2）一个类中的某一个方法有上百或者上千行代码；

（3）代码中没有注释；

（4）为了防止影响历史业务，新需求开发均需将原有部分能复用的代码拷贝后，才能修改。

此时，可以将出现上述问题的微服务进行代码重构。具体来说，可以采用 23 种设计模式、SOLID 原则将包含上千上万行代码的类进行重构。此时，重构后的微服务的上线即可对应上一讲里提到的“第一种重构类型：纯代码重构”。它的架构如下图 2 所示：

图 2：纯代码升级重构

完成纯代码的重构后，在日常维护中，当你发现有以下问题时，可以考虑进行包含存储的重构。

1. 随着业务的发展，微服务的流量由每秒几百的 QPS 上升至上千或上万的 QPS 时，可以将微服务的存储从数据库升级为缓存，以便有效应用业务增长带来的流量。
2. 业务或者运营需要对数据库中的四五张表进行聚合（join）分页查询，而数据库面对这些繁杂查询性能会变得非常差。此时可以将微服务的存储从数据库升级为 ElasticSearch，进而满足多维度的富查询。

上述这两类便为包含存储的重构升级，第一个是从数据库升级到缓存，第二个是从数据库升级到 ElasticSearch。它们的架构如下图 3 所示：

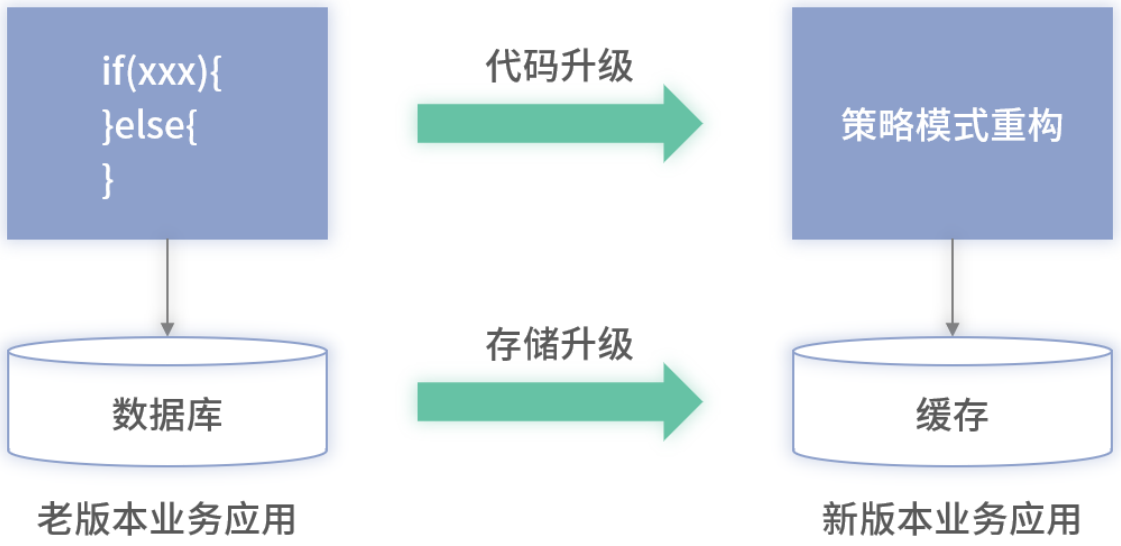

图 3：包含存储的架构升级

烟囱式到平台化的重构升级

完成上述两种重构之后，接下来就需要思考什么时候进行另一种存储类型均为数据库存储，但表结构不同的重构了。

在介绍具体实施步骤前，咱们先来聊聊：什么是烟囱式架构。

这里以即时通信作为讨论案例。在 PC 时代，QQ 在即时通信市场的占有率是绝对领先者，相信你也使用过。从技术的抽象层面来看，QQ 主要提供消息发送和消息接收的功能，消息可以是图片、表情、文字、视频、语音等内容。支撑 QQ 消息发送和接收的简版架构如下图 4 所示：

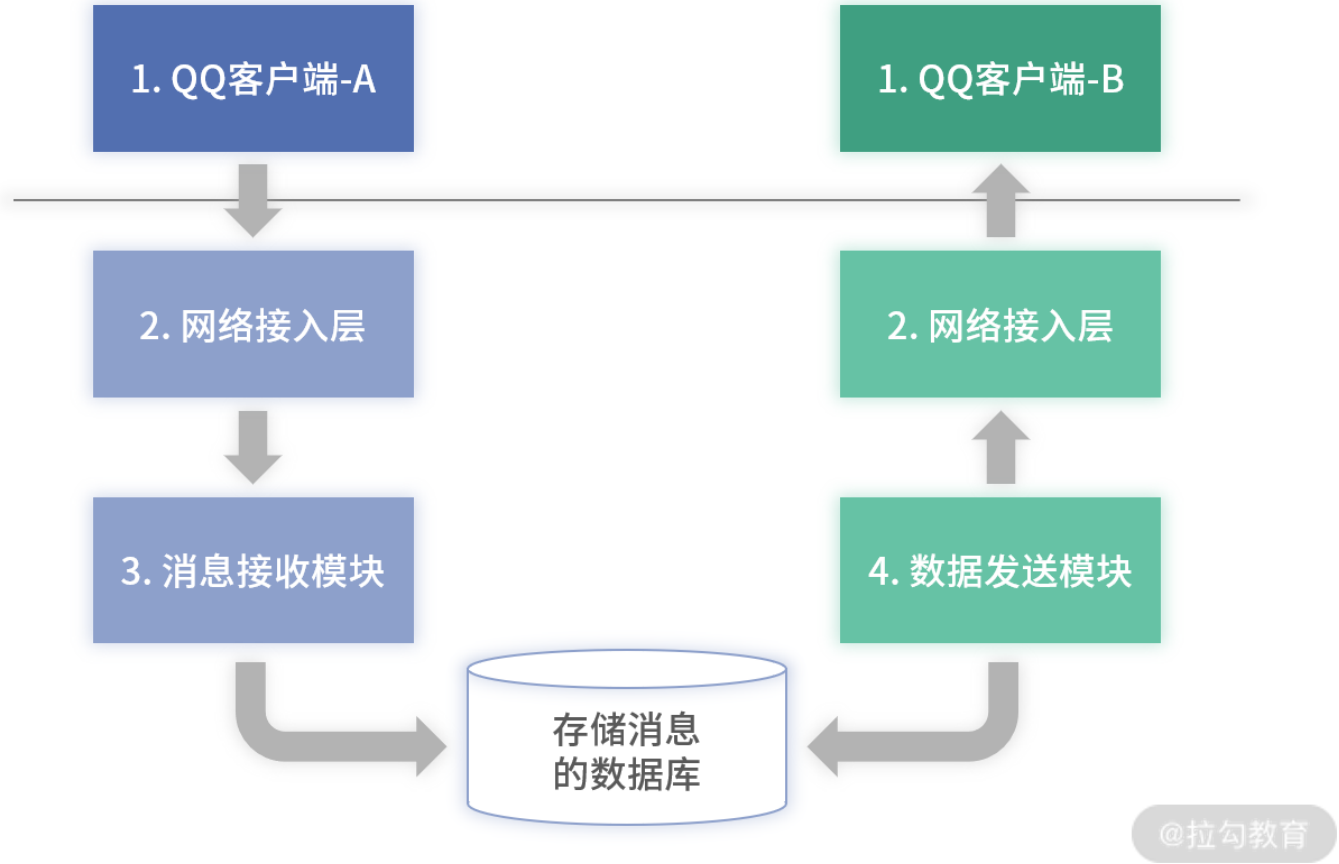

图 4：简版的消息发送和接收架构图

- 图中编号 1 的模块为安装在电脑里的 QQ 客户端，它主要给用户提供可视化的聊天界面，以及发送和接收其他用户的消息。
- 编号为 2 的模块为网络接入层，它主要的作用是维护和客户端的网络连接，负责解析客户端发送到服务端的消息和推送其他用户发送的消息给到指定的客户端。在接收和发送消息时，接入层需基于 QQ 自有的数据协议进行消息内容的解码和编码。
- 编号为 3 的是数据接收模块，它对外提供保存消息的 RPC 接口，并由编号 2 的网络接入层在接收到客户端消息的时候调用。接收模块接收到消息后，会将消息保存至存储中并通知编号 4 的消息发送模块。
- 编号为 4 的是消息发送模块，它接收到通知后，会读取存储中的待发送消息并进行一定的逻辑处理，然后调用网络接入层进行消息的发送。

关于即时通信后续的发展，你应该就有亲身感受了。为了抓住移动互联网的发展浪潮，腾讯又推出了即时通信的王者级应用：微信。从产品上看，微信和 QQ 在定位、应用界面设计、附加功能设计等方面存在差异。但抽象地从技术和核心功能上分析，两者的功能均是给用户提供消息发送和消息接收。因此，在技术实现上，微信研发团队也需要建设和上述图 4 类似的提供消息发送和接收的技术架构，如下图 5 所示：

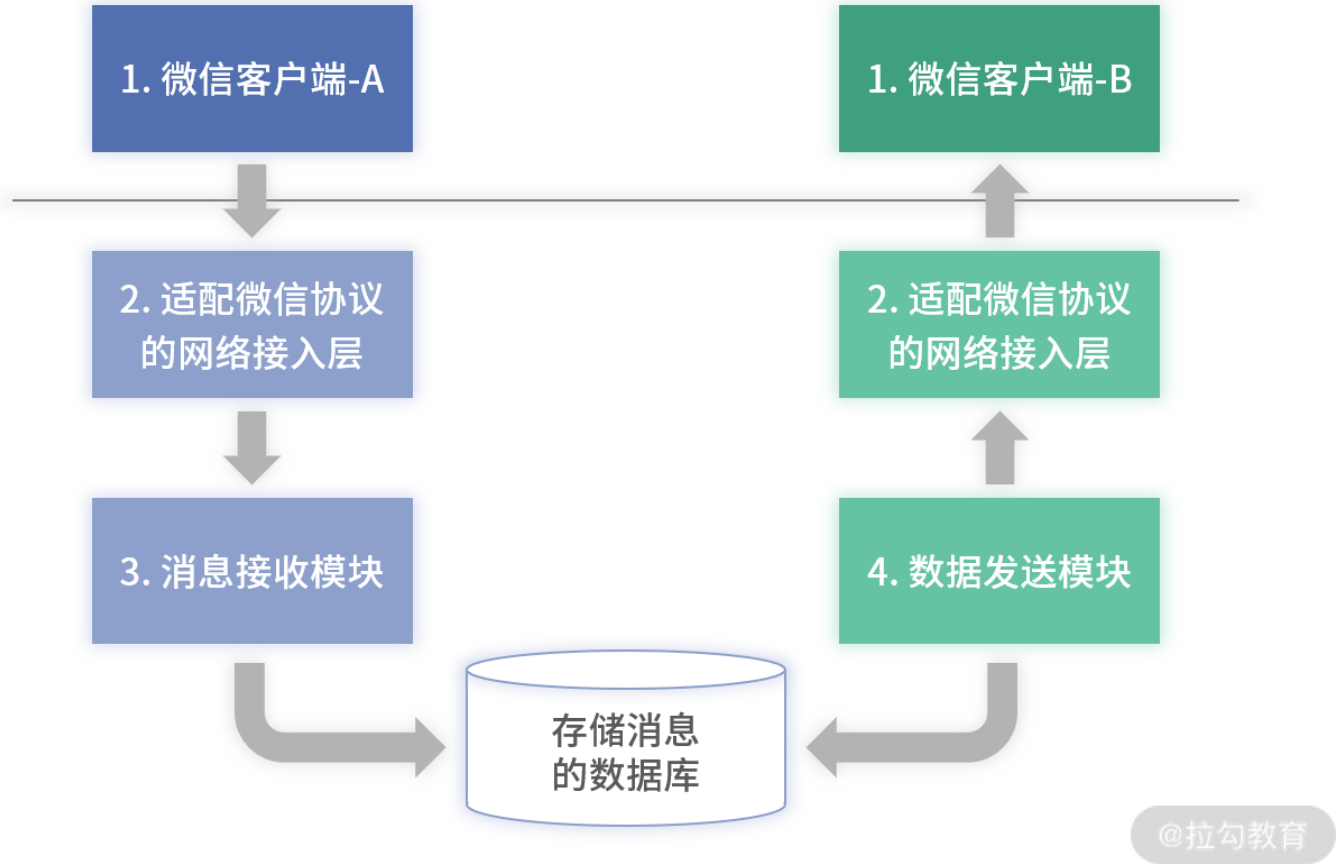

图 5：简版的微信消息发送和接收的架构图

从图 5 中不难发现，除了编号 1 的微信 App 和编号 2 的网络接入层模块与图 4 有差异，其余各模块的功能基本上与图 4 类似。网络接入层之所以有差异，是因为接入层需要进行通信协议的解析，而 QQ 和微信的网络通信协议是根据各自的客户端进行定制的，因此会有格式上的差异。

除了微信外，现在大部分在线游戏也都提供了即时通信的能力。因此，游戏团队也需要按上述类似的架构实现自己的消息发送和接收业务系统。

类似上述介绍的这种模式：即系统架构大体上类似，其中只有个别模块存在差异，但各个研发团队还是从零开始建设全部模块的方式，称为烟囱式架构。为了方便你理解，我把QQ、微信以及游戏语音的架构放在一张图中，如下图 6 所示：

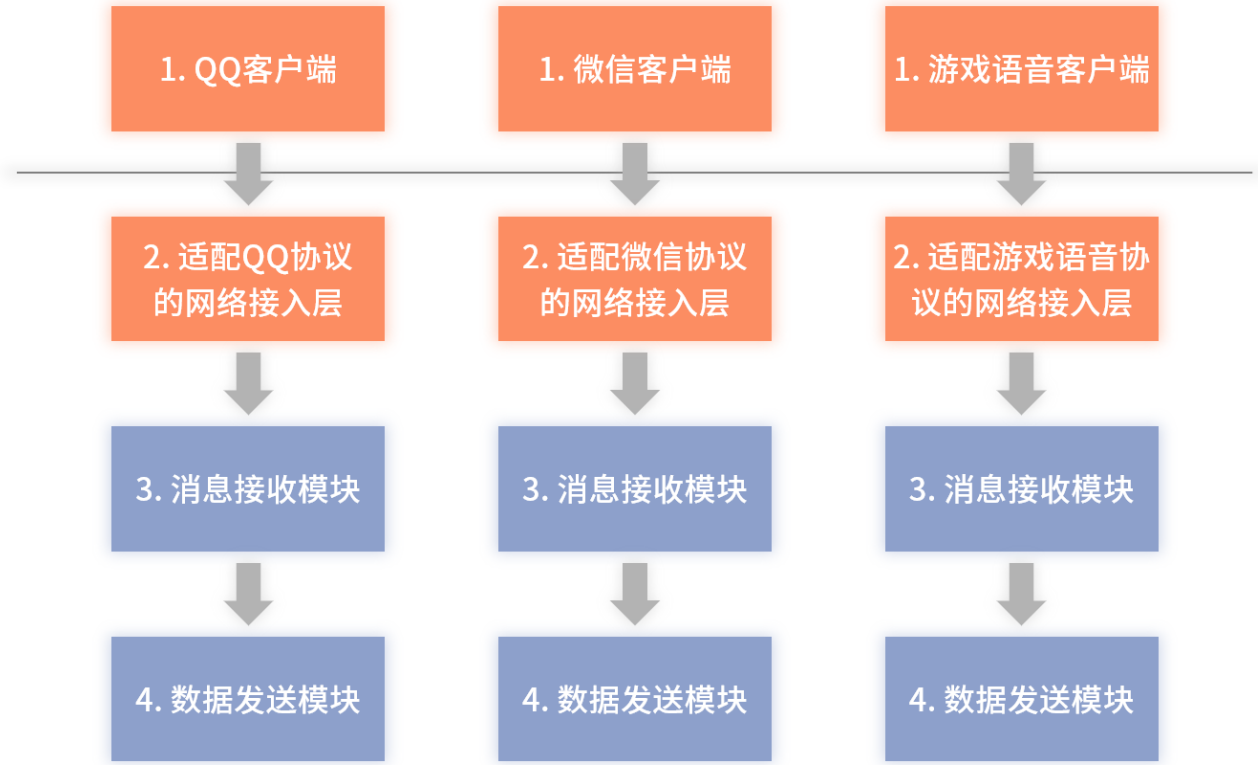

图 6：即时通信的烟囱式架构示意图

从上图可以看出，烟囱式架构是一种象形比喻，各个业务研发团队（如 QQ、微信、游戏语音团队）维护了一个类似烟囱式的、包含重复模块的系统架构。除了即时通信这个案例，还有很多其他会产生烟囱式架构的场景，比如电商，电商除了 PC、App、M 页面版本，现在还有很多购物场景，比如自动贩卖机、微信里的分享链接、小程序等。在实现上，如果他们的研发团队是封闭地进行自研，那么也会产生如下图 7 所示的电商版烟囱式架构。

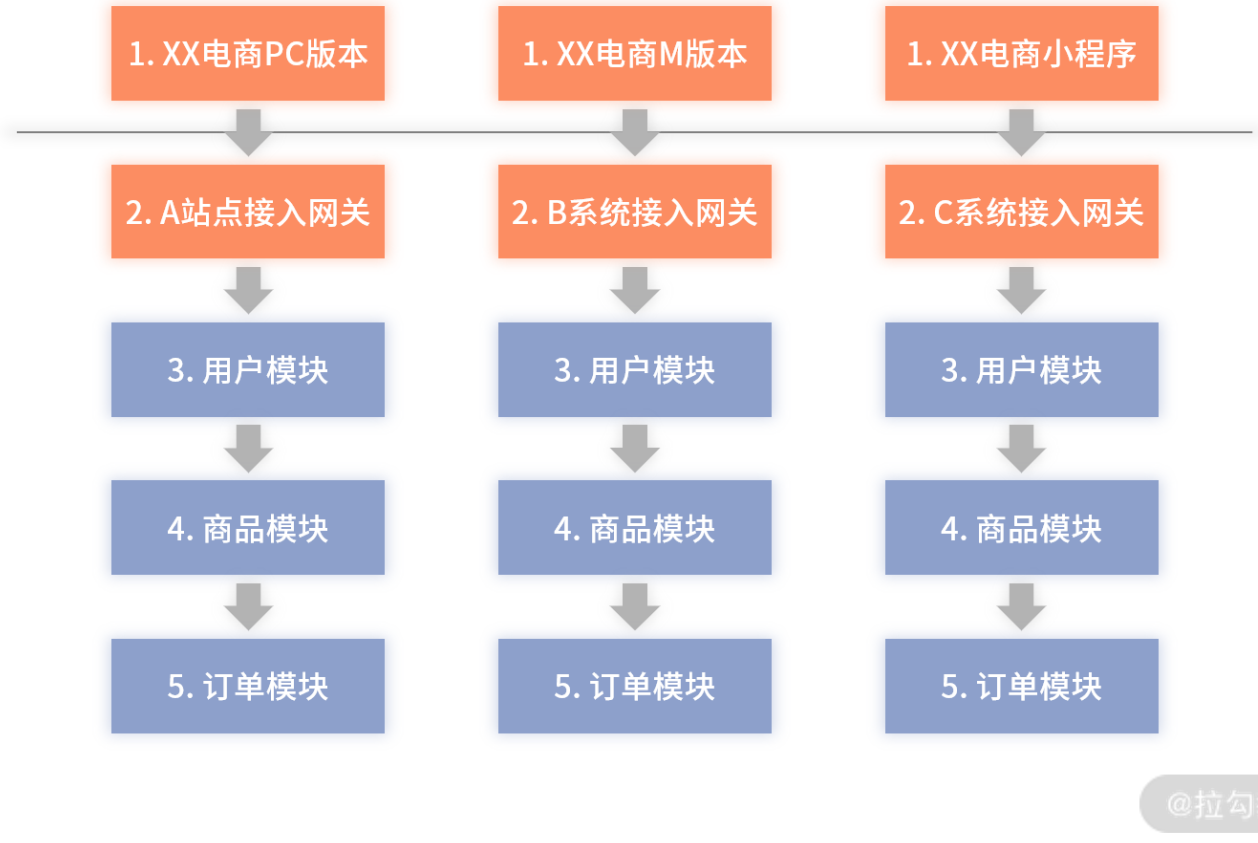

图 7：电商版烟囱式架构示意图

经过前面的分析，烟囱式架构存在的问题其实已经比较明显了，即存在大量的模块重复，进而导致人力重复、成本增加。此时，为了解决这个问题，便可以启动本小节开头提到的升级重构：均为数据库存储，但表结构不同的重构了。

以上述消息接收模块为例，它包含了一个代码进程和对应的消息存储（假设为 MySQL）。为了解决此模块的重复，可以合并 QQ、微信和游戏语音里此模块的代码。同时在前期设计时，各个业务只考虑自己的消息格式，所以它们的数据库的表结构是偏定制的，无法直接被复用，因此在重构时，还需要设计一套全新的、兼容原有三个版本的数据库。此时消息接收模块的重构架构如下图 8 所示：

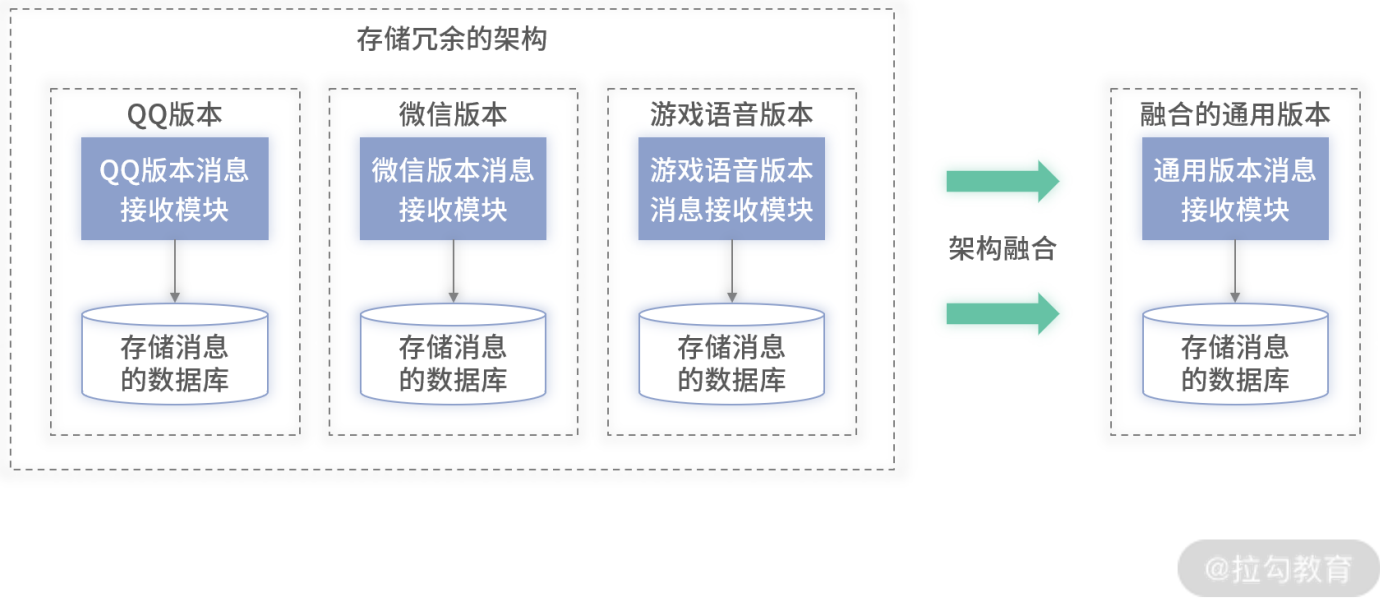

图 8：消息接收模块融合架构图（三个模块+数据库合并成一个）

完成上述模块的重构融合升级之后，消息发送模块也可以进行类似的融合重构。当所有的可复用模块均完成升级重构后，上述三个不同的即时通信软件的架构演化成如下图 9 的形态：

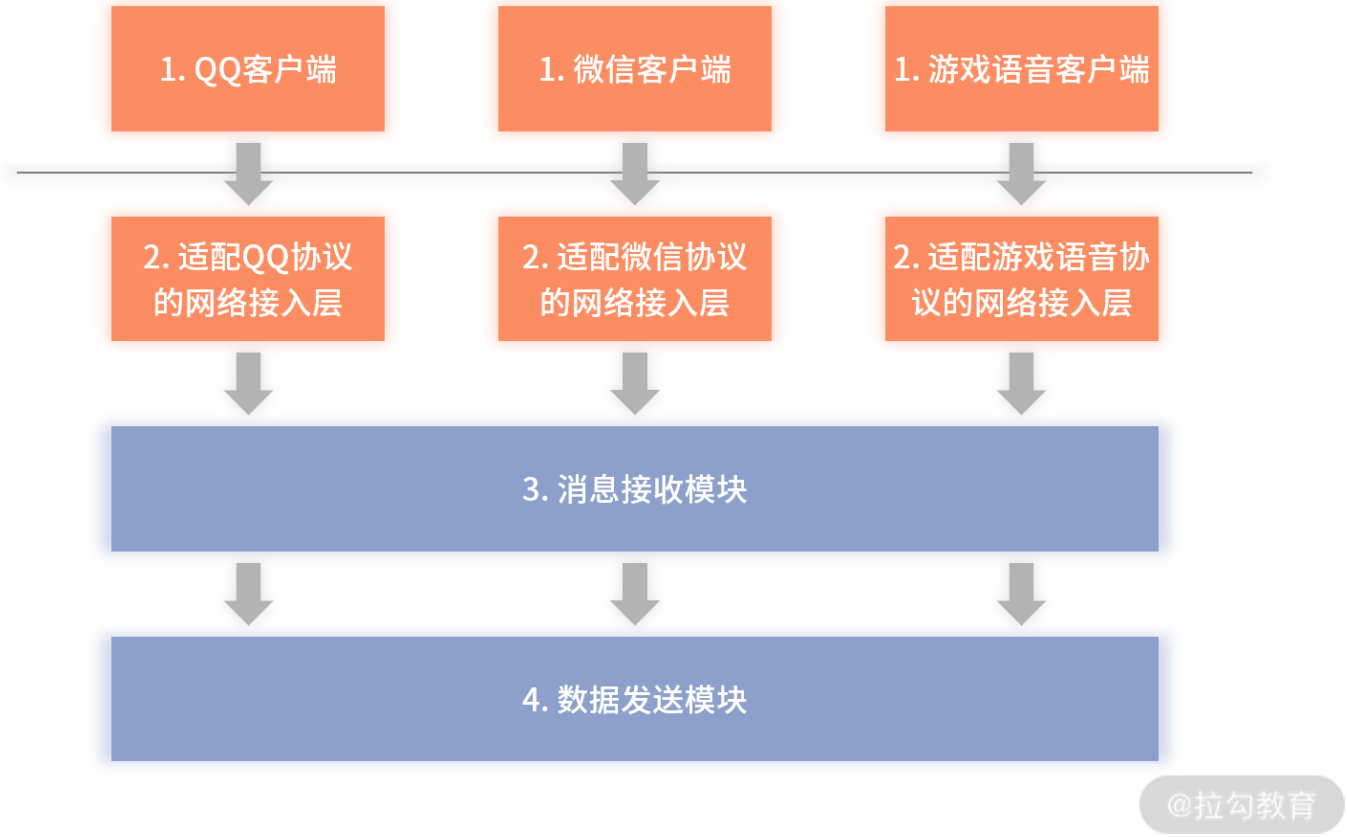

图 9：融合的系统架构

融合后，三款软件有差异的系统模块依然各自维护，但消息接收和发送模块已经融合为一套。此时，从多个模块融合形成的、支持不同类型使用方的模块，称为**平台化模块**。而这个重构过程，有一个高大上的名称：**从烟囱式架构朝着平台化演化**。

平台化到中台化的重构升级

升级重构完成平台化之后，后续三款软件涉及消息接收和发送的新需求，都由平台化模块直接支持。这种需求支撑的模式，看起来十分像这几年兴起的中台化架构，但其实并不是。下面我们具体分析一下，你可以从以下两点来理解。

1. 平台化架构的概念要早于中台化架构。平台化只是将重复模块进行融合，如在平台化之后，未做任何中台化的改造，便不能直接称为中台化模块。
2. 平台化是从降低技术重复的角度出发，从而提升效率，而中台化是在平台化之后，从业务复用的角度出发，进一步提升业务需求的效率。

下面，继续以上一小节的案例作为讨论对象。完成平台化，消灭重复技术之后，如果你遇到以下几种情况，则需要进一步重构，以便完成从平台化到中台化的演化。

1. 融合后的模块代码量庞大、代码中业务逻辑分散。表现就是需求承接时，需要一周时间进行评审，而开发只要一到两天时间，沟通成本巨大。
2. 当出现上述三个之外的新业务场景，融合的平台无法直接支持，而需要大量改造时。
3. 假设 QQ 的某一个需求在平台上开发上线后，微信也提出同样类似的需求，但平台无法直接复用 QQ 的需求，而需要重新开发时。

从平台化到中台化演化升级，可以从业务能力可视化、业务能力在线配置化的方法进行落地改造。

业务能力可视化

仍以上述的数据接收模块为例，可以在平台化之后，将数据接收模块对外的接口流程进行梳理并可视化地展现出来。格式为如下图 10 所示：

图 10：接收数据的接口的可视化展示

> 流程：数据合法性校验 → 图片压缩 → 图片尺寸裁剪 → 图片转存到 CDN → 语音自动识别成文字 → 保存。

上述这个流程图，大多数情况是在需求提出时由产品经理进行绘制，而在代码上线后便不会再实时更新。而中台化之后，需要开发建设一套业务可视化平台，将业务平台中的代码流程可视化地登记到可视化系统中，同时要保证可视化平台能够在业务代码修改后，实时更新相对应的流程。在实现上，编写业务代码时，可以增加一些代码标记，供可视化平台进行自动化扫描，进而识别业务流程，最终更新到可视化平台的显示界面上。

通过将业务能力可视化之后，前面提到的因为平台化融合了太多代码，导致代码量多、业务无法直接从代码中抽取的问题便解决了。因为可视化之后，业务逻辑可以直接在可视化平台上展现出来，业务方和产品经理不需要和研发来回沟通上周的时间来确认需求，可以极大地降低沟通时间，提升效率。

业务能力配置化

在上述图 10 的流程中，可以看到有些是实心的圆圈，有些是空心的圆圈。空心表示代码执行到此流程节点时会直接跳过，而实心表示会执行此流程节点。流程节点是为空心还是实心，是可配置的，此配置功能可以落地在上述介绍的可视化平台里。

上图 10 的可视化、可配置化流程只有一份，但假设微信在保存消息数据时，不希望图片被压缩而用原图保存；而 QQ 在保存消息数据时，不希望图片尺寸被裁剪。此时，如何通过配置化解决这样的需求呢？答案便是：**基于业务身份进行业务流程的配置化**。

业务身份是指给 QQ、微信及游戏语音等每一个复用中台能力的应用，都分配一个全局唯一的名称。在进行配置的时候，按业务身份进行隔离，每一个业务身份都拥有属于自己的上述流程的配置，如下图 11 所示：

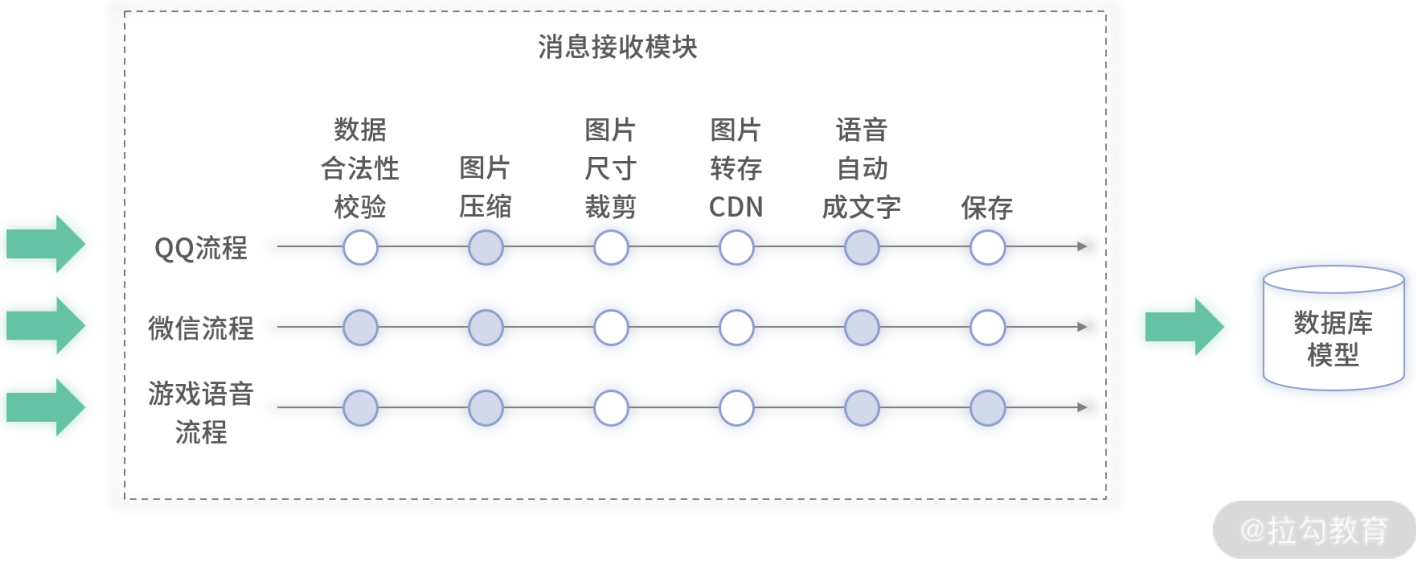

图 11：按业务身份隔离的私有化流程配置图

在执行时，各个即时通信应用在调用保存消息接口时，需传入自己的业务身份标识。对应的中台模块会根据业务身份获取相对应的配置，并按配置去编排属于此业务身份的流程。

再回顾一下本小节开头提出的平台化架构会遇到的问题。

1. 再来一个新的聊天应用时，无法快速、直接复用已有能力。采用配置化后，可以给新的聊天应用配置一个业务身份，同时基于此应用的需求，去配置它需要使用的业务节点。
2. QQ 先提出的某一个业务需求并开发上线后，后续微信也想要此功能，但无法快速直接复用。在完成上述业务功能可视化、配置化的基础上，当 QQ 先提出的需求上线后，可视化的工具会将此新功能直接更新到上述流程节点里。只是在 QQ 对应的业务身份的配置里，此新加入的节点为实心。而其他不使用此新功能的业务身份里，此节点为空心。当后续微信需要使用此新功能时，直接将此节点勾选为实心，便可直接复用。

至此，从平台化到中台化的重构升级便具备基本雏形。当前中台化的建设理论还处于初期，有很多种探索的实现方式，但万变不离其宗，它的核心仍然是：**在面对不断出现的新的业务场景和形态时（如电商里新出现的社区购等），中台需要快速地复用已有能力，去满足业务新建站点或不断扩宽业务边界的诉求**。

总结

罗马不是一日建成的，系统建设也是一样，它是随着业务的发展不断演化而来的。当业务体量较小且没有类似像 QQ 和微信的多个前台应用时，没有必要在建设初期就采用平台化或中台化的建设方案。因为它们的建设人力成本和消耗的机器资源也更高。

一个系统在建设时，假如预期未来的三到五年的用户量并不会增长太大，可以先采用烟囱式的架构，快速地满足业务需求。当发展到一定体量后，再发起从烟囱式到平台化及中台化演化即可。毕竟能够发展到百万、千万用户体量的系统是少数，所有的系统都提前建设会存在较大可能的成本浪费。

最后，我再给你留一个讨论话题：当前你所负责的系统处在什么样的阶段，是烟囱式、平台化或中台化的架构吗？它存在什么样的问题，你觉得是否有必要准备启动升级计划了。欢迎你在留言区说出你的想法，我们一起讨论。

这一讲就到这里，感谢你学习本次课程，下一讲是本专栏的最后一篇内容，我想和你聊聊关于程序员发展的话题：抓住本质，是成为技术专家的不二法则。

最后，我邀请你为本专栏课程进行结课评价，因为你的每一个观点都是我和拉勾教育最关注的点。点击链接，既可参与课程评价 。编辑会随机抽 5 位同学送精美礼品喔。