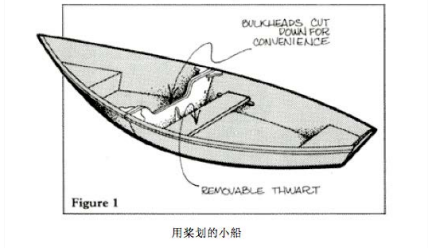
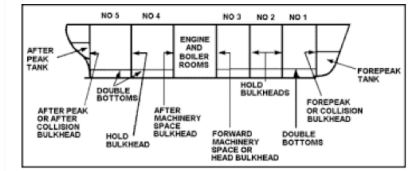
弹力设计篇之“隔离设计”

2018-02-22 陈皓

隔离设计对应的单词是Bulkheads，中文翻译为隔板。但其实，这个术语是用在造船上的，也就是船舱里防漏水的隔板。一般的船无论大小都会有这个东西，大一点的船都会把船舱隔成若干个空间。这样，如果船舱漏水，只会进到一个小空间里，不会让整个船舱都进水而导致整艘船都沉了，如下图所示。



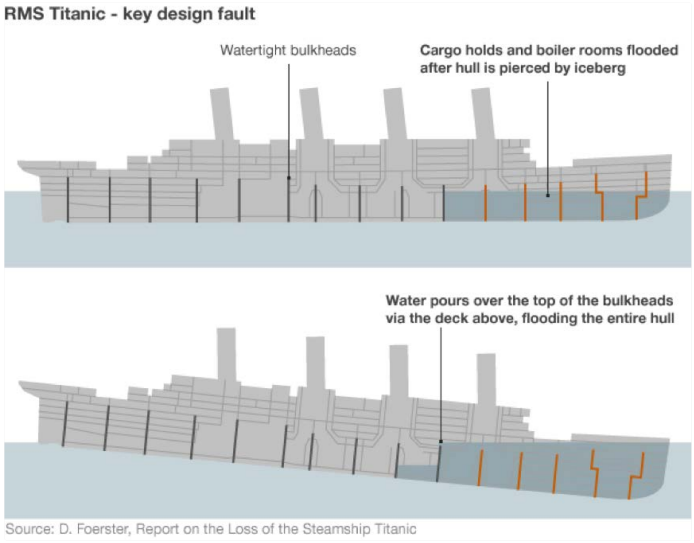
用桨划的小船



大一点的运输船

我们的软件设计当然也“漏水”，所以为了不让这个“故障”蔓延开来，需要使用“隔板”技术，来将架构分隔成多个“船舱”来隔离故障。

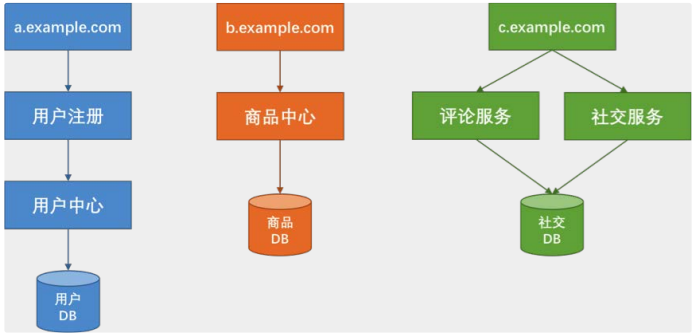
多扯一句，著名的泰坦尼克号也有Bulkheads设计，然而其设计上有个缺陷。如下图所示，当其撞上冰山漏水时，因为船体倾斜，导到水漫过了隔板，从而下沉了。



在分布式软件架构中，我们同样需要使用类似这样的技术来让我们的故障得到隔离。这就需要我们对系统进行分离。一般来说，对于系统的分离有两种方式，一种是以服务的种类来做分离，一种是以用户来做分离。下面具体说明一下这两种方式。

按服务的种类来做分离

下面这个图中，说明了按服务种类来做分离的情况。



上图中，我们将系统分成了用户、商品、社区三个版块。三个板块分别使用不同的域名、服务器和数据库，做到从接入层到应用层再到数据层三层完全隔离。这样一来，在物理上来说，一个版块的故障就不会影响到另一版块。

在亚马逊，每个服务都有自己的一个数据库，每个数据库中都保存着和这个业务相关的数据和相应的处理状态。而每个服务从一开始就准备好了对外暴露。同时，这也是微服务所推荐的架构方式。

然而任何架构都有其好和不好的地方，上面这种架构虽然在系统隔离上做得比较好，但是也存在以下一些问题。

- 如果我们需要同时获得多个版块的数据，那么就需要调用多个服务，这会降低性能。注意，这里性能降低指的是响应时间，而不是吞吐量（相反，在这种架构下，吞吐量可以得到提高）。

对于这样的问题，一般来说，我们需要小心地设计用户交互，最好不要让用户在一个页面上获得所有的数据。对于目前的手机端上来说，因为手机屏幕尺寸比较小，所以，也不可能在一个屏幕页上展示太多的内容。

- 如果有大数据平台，就需要把这些数据都抽取到一个数据仓库中进行计算，这也增加了数据合并的复杂度。对于这个问题，我们需要一个框架或是一个中间件来对数据进行相应的抽取。
- 另外，如果我们的业务逻辑或是业务流程需要跨版块的话，那么一个版块的故障也会导致整个流程走不下去，同样会导致整体业务故障。

对于这个问题，一方面，我们需要保证这个业务流程中各个子系统的高可用性，并且在业务流程上做成Step-by-Step的方式，这样用户交互的每一步都可以保存，以便故障恢复后可以继续执行，而不是从头执行。

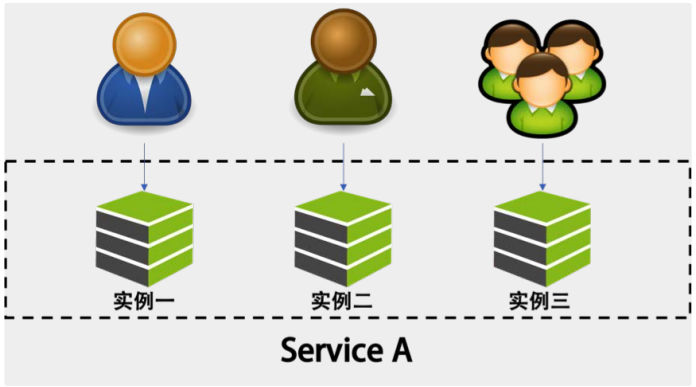
- 还有，如果需要有跨版块的交互也会变得有点复杂。对此我们需要一个类似于Pub/Sub的高可用的并可以持久化的消息订阅通知的中间件来打通各个版块的数据和信息交换。
- 最后还会有在多个版块中分布式事务的问题。对此，我们需要“二阶段提交”这样的方案。在亚马逊中，使用的是Plan – Reserve – Commit/Cancel 模式。

也就是说，先做一个plan的API调用，然后各个子系统reserve住相应的资源，如果成功，则Commit；如果有一个失败，则整体Cancel。这其实很像阿里的TCC – try confirm/cancel。

可见，隔离了的系统在具体的业务场景中还是有很多问题的，是需要我们小心和处理的。对此，我们不可掉以轻心。根据我的经验，这样的系统通常会引入大量的异步处理模型。

按用户的请求来做分离

下图是一个按用户请求来做分离的图示。



在这个图中，可以看到，我们将用户分成不同的组，并把后端的同一个服务根据这些不同的组分成不同的实例。让同一个服务对于不同的用户进行冗余和隔离，这样一来，当服务实例挂掉时，只会影响其中一部分用户，而不会导致所有的用户无法访问。

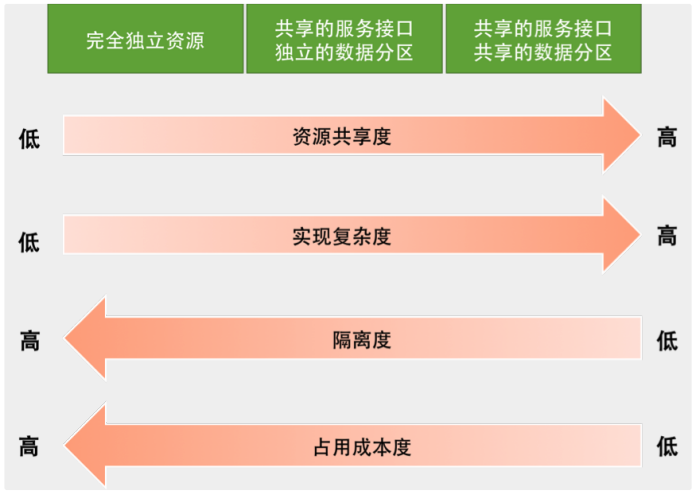
这种分离和上面按功能的分离可以融合。说白了，这就是所谓的“多租户”模式。对于一些比较大的客户，我们可以为他们设置专门独立的服务实例，或是服务集群与其他客户隔离开来，对于一些比较小的用户来说，可以让他们共享一个服务实例，这样可以节省相关的资源。

对于“多租户”的架构来说，会引入一些系统设计的复杂度。一方面，如果完全隔离，资源使用上会比较浪费，如果共享，又会导致程序设计的一些复杂度。

通常来说多租户的做法有三种。

1. 完全独立的设计。每个租户有自己完全独立的服务和数据。
2. 独立的数据分区，共享的服务。多租户的服务是共享的，但数据是分开隔离的。
3. 共享的服务，共享的数据分区。每个租户的数据和服务都是共享的。

这三种方案各有优缺点，如图所示。



通过上图，可以看到：

- 如果使用完全独立的方案，在开发实现上和资源隔离度方面会非常好，然而，成本会比较高，计算资源也会有一定的浪费。
- 如果使用完全共享的方案，在资源利用和成本上会非常好，然而，开发难度非常大，而且数据和资源隔离非常不好。

所以，一般来说，技术方案会使用折衷方案，也就是中间方案，服务是共享的，数据通过分区来隔离，而对于一些比较重要的租户（需要好的隔离性），则使用完全独立的方式。

然而，在虚拟化技术非常成熟的今天，我们完全可以使用“完全独立”（完全隔离）的方案，通过底层的虚拟化技术（Hypervisor的技术，如KVM，或是Linux Container的技术，如Docker）来实现物理资源的共享和成本的节约。

隔离设计的重点

要能做好隔离设计，我们需要有如下的一些设计考量。

1. 我们需要定义好隔离业务的大小和粒度，过大和过小都不好。这需要认真地做业务上的需求和系统分析。
2. 无论是做系统版块还是多租户的隔离，你都需要考虑系统的复杂度、成本、性能、资源使用的问题，找到一个合适的均衡方案，或是分布实施的方案尤其重要，这其中需要你定义好要什么和不要什么。因为，我们不可能做出一个什么都能满足的系统。
3. 隔离模式需要配置一些高可用、重试、异步、消息中间件，流控、熔断等设计模式的方式配套使用。
4. 不要忘记了分布式系统中的运维的复杂度的提升，要能驾驭得好的话，还需要很多自动化运维的工具，尤其是使用像容器或是虚拟机这样的虚拟化技术可以帮助我们更方便地管理，和对比资源更好地利用。否则做出来了也管理不好。
5. 最后，你需要一个非常完整的能够看得到所有服务的监控系统，这点非常重要。

小结

好了，我们来总结一下今天分享的主要内容。首先，我从船体水密舱的设计，引出了分布式系统设计中的隔离设计。然后我介绍了常见的隔离有两种，一种是按服务种类隔离，另一种是按用户隔离（即多租户）。下篇文章中，我们讲述异步通讯设计。希望对你有帮助。

也欢迎你分享一下你是如何为分布式系统做隔离设计的。

文末给出了《分布式系统设计模式》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

- 弹力设计篇
  - 认识故障和弹力设计
  - 隔离设计Bulkheads
  - 异步通讯设计Asynchronous
  - 幂等性设计Idempotency
  - 服务的状态State
  - 补偿事务Compensating Transaction
  - 重试设计Retry
  - 熔断设计Circuit Breaker
  - 限流设计Throttle
  - 降级设计degradation
  - 弹力设计总结
- 管理设计篇
  - 分布式锁Distributed Lock
  - 配置中心Configuration Management
  - 边车模式Sidecar
  - 服务网格Service Mesh
  - 网关模式Gateway
  - 部署升级策略
- 性能设计篇
  - 缓存Cache
  - 异步处理Asynchronous
  - 数据库扩展
  - 秒杀Flash Sales
  - 边缘计算Edge Computing

来

2018-05-20

我们目前系统中采用隔离的点包括：
1、服务集群隔离，我们可以配置不同的请求访问不同的服务集群，我们通过服务别名来区分
2、数据存储隔离，包括数据库隔离、缓存集群隔离。数据库隔离一般通过分库分表，读写分离
3、线程池隔离，在同一个应用中，不同的任务处理通过线程池隔离
4、网络带宽隔离

暂时想到这么多，我理解隔离的本质是当系统出尽现故障时，尽可能的将故障影响范围降到最低。

北极点

2018-03-22

隔离设计感觉是一个随着系统逐步进化，业务逐渐成熟的前提情况下诞生出来的模式。特别是多租户的设计！我之前的工作当中要是早了解或者思考下这种设计可能就会在维护现有的系统时考虑设计了，或者也会给技术管理层领导提建议了！读这篇文章很有感触。

---

shufang

2018-03-02

多租户的实例是指请求层服务层数据层的完全隔离吗？

作者回复

2018-03-02

多租户的隔离有三种方案，我在文中说了，请仔细阅读。