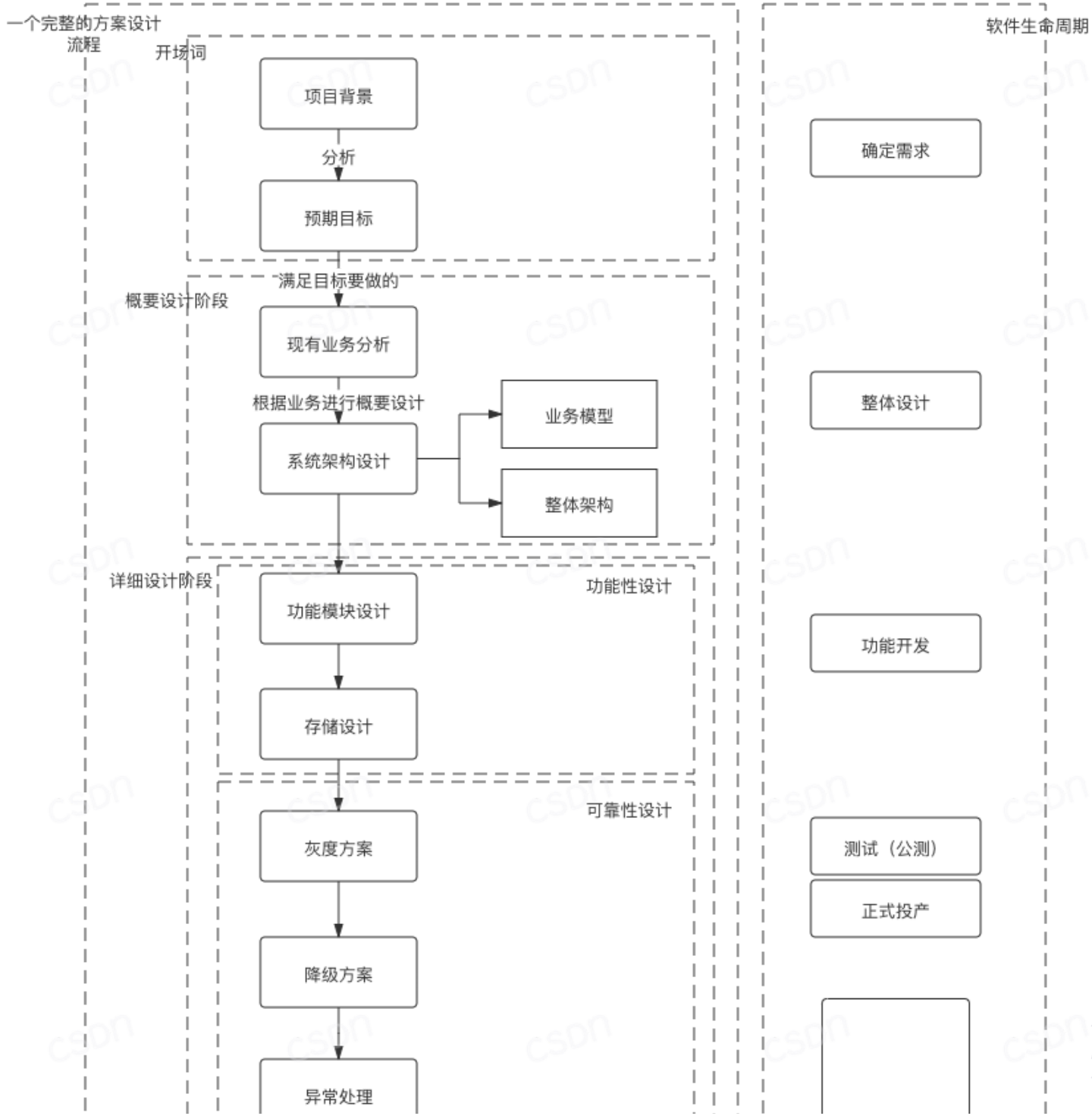
如何写一篇可实施的技术方案?

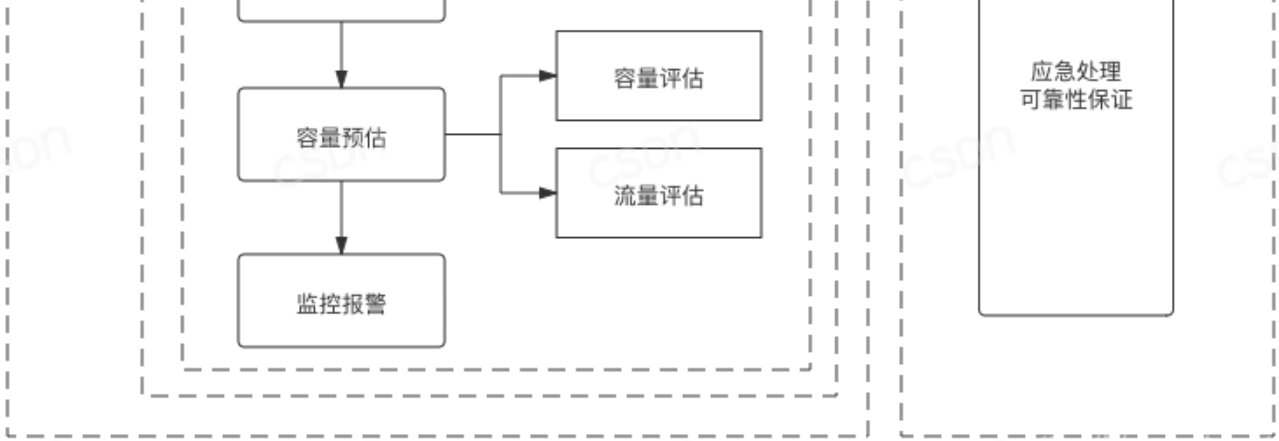
bd7xzz 于 2020-05-11 23:28:27 发布

为何要写这篇博文?

在日常开发中，老大经常要求我们给出一个完善并合理的技术方案之后才能进行开发。并且要求技术方案一定要细，要重点覆盖监控、异常处理、灰度、降级方案。同时要注重边界处理。最初，我的技术方案写的很粗，也没有理解老大说的边界处理到底是怎么一回事。于是乎，辛辛苦苦写了一周的方案，就会在技术方案评审的时候直接打回重做，甚至多次打回。

不过还好，在经历过几次大项目的方案设计后，我的方案设计越来越完善，直到最后老大非常认可并在组内进行参考。随着我的方案设计逐渐完善，也逐渐发现，不但编码效率越来越高，编码时思维更加清晰，而且方案中的每一个模块都贯穿了整个软件生命周期。





在这里我会总结以往方案设计，给出一个方案设计模板，并给出模板中每一个模块的具体说明和案例，同时介绍如何通过领域驱动设计的思想拆分业务逻辑引入到这个设计模板中。
这一切看起来可能很枯燥乏味，但如果你认真读下去我想会对你有所帮助。
如果你在一个垃圾公司做的项目从来没上过线，或者上线就崩溃，可以认真看看，一个大流量、高稳定性的系统是如何实现的。
如果你有更好的方案设计，欢迎留言讨论，或者喷我。让我能够更加完善我的设计，谢谢！

一个错误的方案设计

在说如何编写一个好的技术方案之前，先说说一个错误的方案。
其实要说错误的方案设计是什么样子的，是很难界定出来的。错误可能有很多点，导致的后果也是不同的。比如：

1. 技术方案设计的比较粗，给出了架构设计图，但没有表述清楚架构层次、模块关系、边界等，可能表面上看着没什么，但是当真正开始实施起来，发现：
   a. 代码职责混乱（比如 controller 层写业务代码）
   b. 模块关系混乱（比如用户服务中调用了订单服务用来显示用户买了什么订单，而订单服务又调用了用户服务显示订单中的收获地址）
   c. 边界混乱（用户服务中写了订单服务的代码，订单服务中写了用户服务的代码）
2. 技术方案没有横向对比，没有对以往的业务进行 review，没有对技术选型进行业内调研和对比。这时候会发现：
   a. 修改的代码导致以往的业务无法兼容，出现不可预知的 bug
   b. 用到的基础技术在某个环节无法支撑你的变更，只能重新选型
3. 没有异常处理方案，在代码开发完成后，虽然有 QA 保证质量。但是实际生产情况总会出现异常情况（网络抖动、用户违法输入、中间件或底层存储崩溃、依赖服务挂掉），在上线后会发现：
   a. 网络抖动数据写入失败，业务请求的数据丢失，要赔偿用户的损失
   b. 用户违法输入，产生了 sql 注入，一切凉凉

c. 中间件或存储崩溃，业务彻底中断，一切凉凉
d. 依赖服务挂掉，业务彻底中断，一切凉凉

4. 没有灰度方案，没有降级方案，异常没有报警，在代码跑了一段时间会发现：
   a. 没有灰度，在某种场景 QA 没有覆盖到，线上用户操作突然触发了，导致大量数据错误甚至全量数据错误
   b. 没有降级，随着用户数据逐渐增多，虽然底层存储性能已经足够高了，但还会出现慢 sql、大表等问题，一个耗费很高性能的 sql 查询拖死整个业务线，最终业务不可用
   c. 异常没有报警，光有日志输出不够，大量的日志会把某一个小问题的 error 日志覆盖掉，导致小问题被扩大
5. 没有存储设计，或者给出的存储结构很粗，在开发时再进行补救，会发现：
   a. 存储结构越加越多，最后逐渐不可控
   b. 发版时，一些建表语句后面加的字段忘记加上去了，导致发版失败或业务请求失败（这个问题也可能出现在消息队列、动态配置文件或配置中心）
6. 没有容量预估，没有流量预估，很多人在测试阶段不会进行流量上的压测和容量上的压测，上线之后会发现：
   a. 老板拉来投资了，砸钱引流服务撑不住了，横向扩容达到瓶颈（**注意：不是加机器就能解决问题的，不信？自己压测试试**）
   b. ugc 数据太多了，数据库很快就被打满了，打慢了，没有分库分表方案
   c. 一个对象产生的太大了，流量多了内存被打爆了（实际发生过！800k 的字符串 100 个线程就触发频繁 gc，引用 hang 死！）

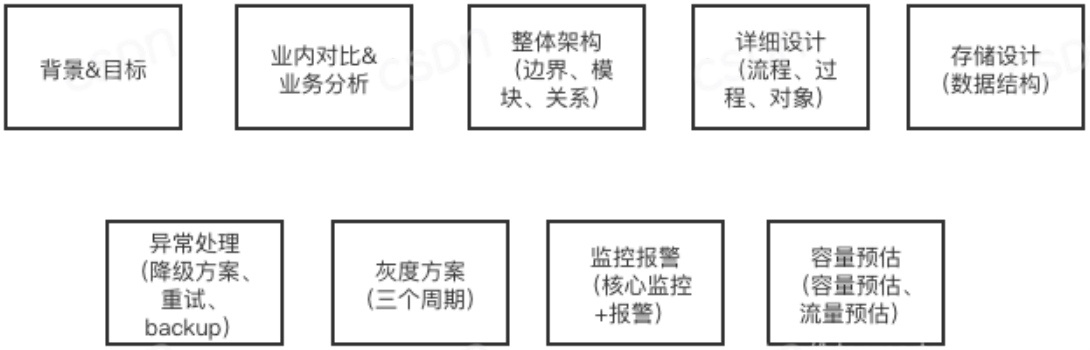
重点是什么?

好吧，上面说了那么多问题，接下来应该说说如何写好一个方案设计了。直接敲黑板说重点！个人认为一个好的方案设计就是为了完全避免上面的那些问题而设计的。所以重点就在于：

1. 技术方案要细，尤其注重**模块（职责明确的模块或者组件）、关系（组件间明确的关联关系）、边界（约束和指导原则）**。可以借助领域驱动设计（DDD）利器进行设计，本质上领域驱动设计的整个思想应该贯穿于整个方案设计中。
2. 要对比业内的方案和选型，分析以往业务产生的效果和方案的优缺点。确定背景和目标，用于指导整个方案设计，防止方案的腐化、跑偏。
3. 业务功能模块要进行详细设计，给出流程图、领域模型建模过程、领域对象等。每个模块的详细设计中可包含伪代码、特殊问题说明、注意事项。
4. 考虑所有可能出现的异常情况，读流程是否可降级为不展示，写流程是否可有备选通道支持（比如 MQ 和接口双通道）。对于抖动、依赖服务短时间崩溃要考虑重试机制。对服务内部异常，要考虑给调用者一个合理可接受的返回值，调用者可根据实际情况进行重试、异常提示等。对核心链路的异常要有及时报警（短信、邮件、电话、IM 消息等）。
5. 永远考虑最差情况，要知道什么是墨菲定理。所以要有降级方案（自动降级、人工降级），当然降级不是万事大吉的，要考虑降级是否对主流程产生影响，对数据是否产生影响，区分有损降级和无损降级。若有损产生了数据异常，则要考虑数据的修复。
6. 实现灰度，还是永远考虑最差的情况，灰度能够控制你的异常影响范围最低，出现异常你修复数据也会最少。方案要给出灰度周期，经验上灰度周期可分为：上线一周（20% 的流量）、上线第二周（50% 的流量）、上线第三周（100% 的流量）。

7. 给出存储设计，要涵盖所有存储（ES、MySQL、Redis、文件系统、CDN 等等），给出详细的存储结构，如果你的存储结构给的很明细，通常在代码开发时不会变动很大，甚至没有变动。
8. 给出流量预估和容量预估，预估你的存储占用大小（内存、持久化）、预估你的 MQ 高峰期流量、预估你的接口高峰期流量、预估你的一次请求 Jvm 内存对象占用的大小。如何预估 Jvm 内存对象的大小？改天写个 blog 说说这事。

好了，这些都是重点。不过你会发现，8 个重点里 4、5、6、7、8 都是在考虑各种异常情况下的各种补救措施，所以一个好的技术方案设计就是考虑到所有的异常情况。同样，**一个技术人，我觉得不是看看 “阿里 p8” 在今日头条和 QQ 群里发的课程内容就能修炼成精的**，经历了 N 多次的线上事故总结，并总结如何避免事故的发生，才是技术人应该做的。



拿模板说说

这里，给出一个我总结的技术方案设计模板，我会拿这个模板来说说如何做一个正确的方案设计。这个模板包含了上面说的所有重点项。如果你能够理解上面我所说的一切，利用好这套模板，拿出去吹 nb 是没什么问题的！

模板如下（忽略格式）：

1. 背景

详细描述项目背景，简单说明以往业务带来的效果。给出为何要进行本次项目迭代的原因。

2. 目标

列出预计产出的业务指标（如：提升用户转化率 30%）和技术指标（如：支撑多大 QPS）。

3. 现有业务分析

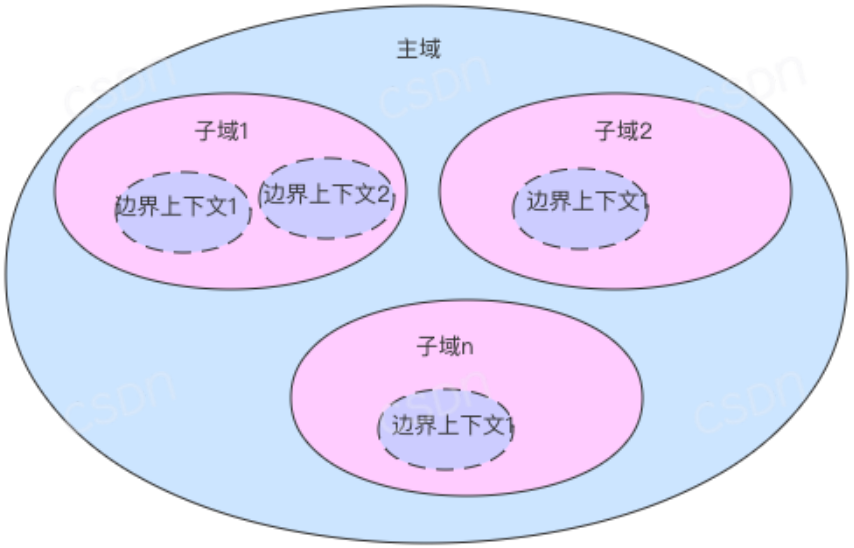
现有业务分析中通常给出原有业务（本次基于原有业务迭代）或对比业内业务和技术的选型（新技术或原有业务迭代）。可以用表格列出原有业务和技术产生的效果、不足点，也可以用表格给出对比行业内多个现有方案的对比，写出每个方案的优缺点，以及是否与自己业务实际情况相匹配。
同时这里需要给出通用语言。

4. 系统整体架构

微服务提倡使用领域驱动设计的方式实现整体架构设计。将业务系统划分出核心域、支撑域、通用域。
核心域：通常要投入的成本比较大，在做技术方案设计时可以主要强调核心域。
通用域：可以引用通用域，在技术方案中进行简要的说明，给出如何依赖通用域，描述涉及到的调用关系以及如何调用的。
支撑域：通常项目的开发也会涉及到围绕着核心域的支撑域，所以可以和核心域一样进行设计。如果支撑域是现成的，同样给出依赖关系以及如何调用。

业务模型

业务模型是领域驱动设计的核心，也是战略设计时必须圈定出来的。可以画图给出子域、边界上下文、聚合，可以用表格标记出事件风暴规划出的域、聚合根、边界上下文、驱动类型。从业务边界和业务角色两种给出模型图。



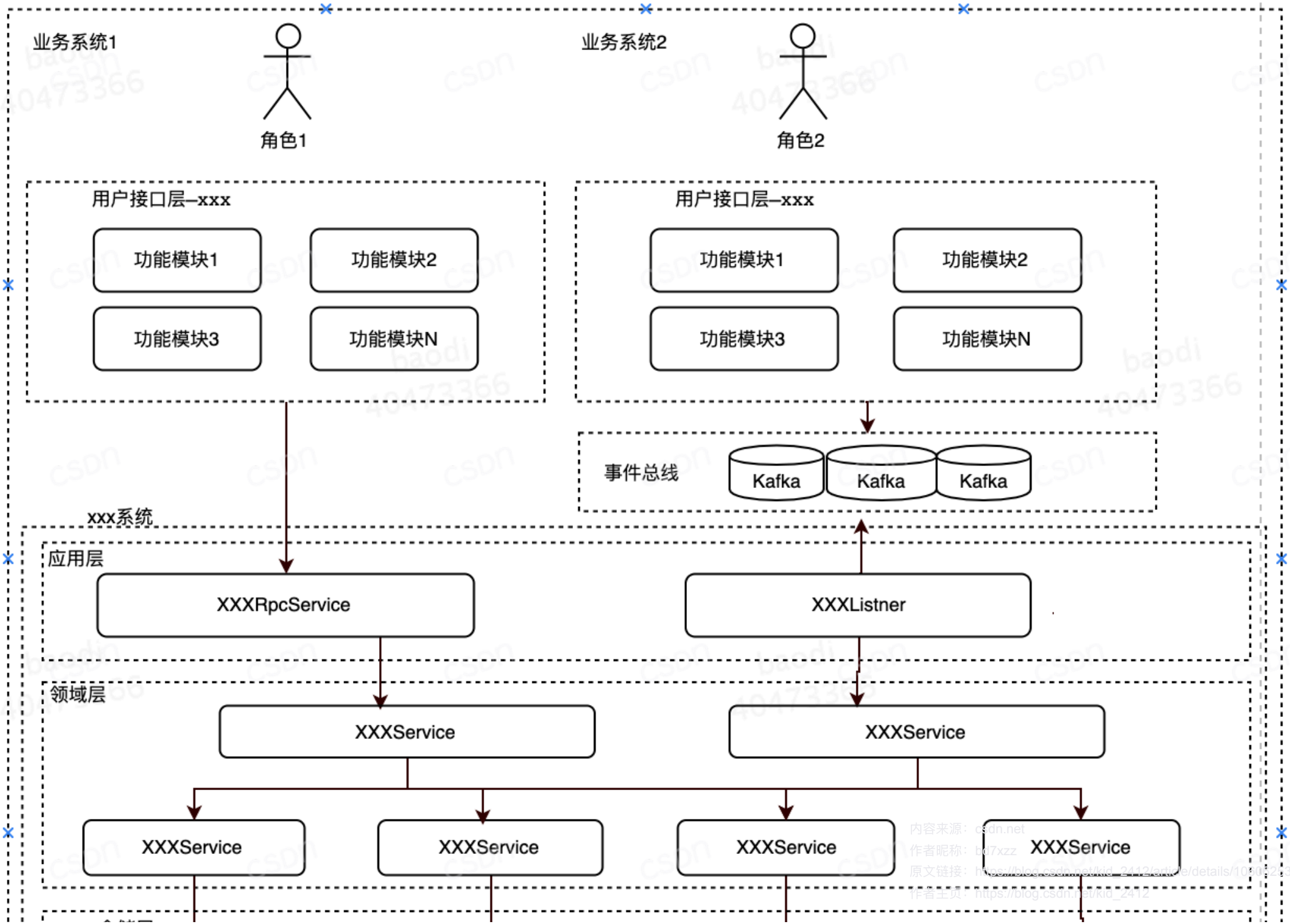
域	聚合根	驱动类型（消息 / 接口）	边界上下文

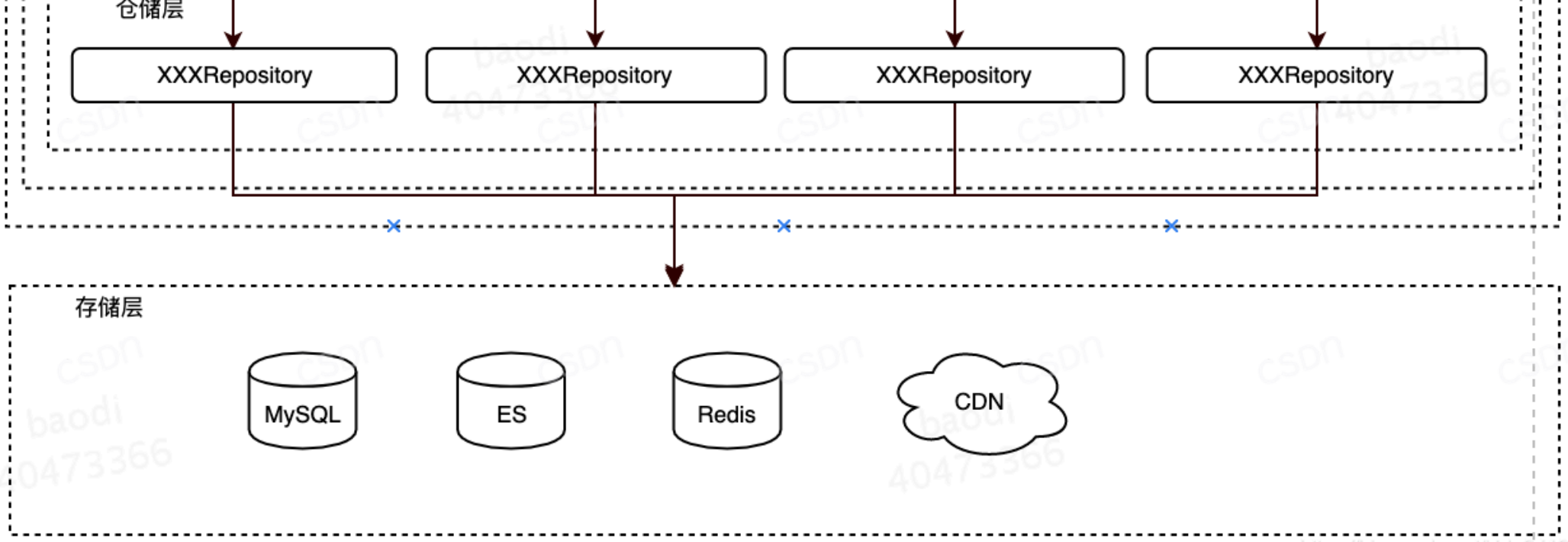
层	聚合	对象	类型	依赖对象	包名	类名	方法名

整体架构

上面给出了业务模型，划分出各层边界，这里就可以通过一张图给出领域模型经典的四层模型，用箭头画出上下层的依赖关系。画图时用不同的图标和颜色区分出层次、模块、角色、基础服务。如果能够区分出读写事件和命令，那么可以用 CQRS 的模型做。
画出图后，用适当的文字描述出关键模块、依赖关系、约束边界。

如果你是在修改以前的架构，这里给出修改之前的架构和修改之后的对比图。





5. 详细设计

进入详细设计，就要给出详细的流程图，并用文字描述出流程。用表格给出实体、值对象、域内服务，架构层面做了 CQRS 的话，这里明确出给出 CQRS 的职责。

xxx 实体 / 域内服务

属性	类型	备注

方法名	出参	入参

xxx 值对象

属性	类型	备注

如果你是流程改造，要给出原有流程和现有流程的对比。若原有流程产生过问题，这里可以给出问题现象、原因，来证明新的流程进行优化原有流程。
对于关键部分可以给出伪代码，关键说明要用比较明显的字体样式。

6. 存储设计
用表格的形式给出存储结构（包括 MySQL、ES、Redis 等），说明 Schema、字段类型、默认值、描述信息等。

表 / 索引 xxx

字段名	字段类型	默认值	是否可空	备注

7. 灰度方案
如果你的公司有灰度条件的话（业务有一定流量），用表格给出灰度范围、配置灰度开关的参数、灰度周期。灰度的三个周期：
a. 第一周：20% 流量，比如某些二三线城市
b. 第二周：50% 流量，比如一些一线城市 + 二线城市
c. 第三周：100% 流量，比如全国
灰度可以让你在及时出现 bug 的情况下损失降到最低。

模块	灰度维度	配置变更	灰度周期（1）	灰度周期（2）	灰度周期（3）

8. 降级方案
及时通过灰度跑了很长时间，也不能 100% 保证你的代码是没有问题的。通常一个业务刚上线很稳定，但是随着时间的推移问题逐渐暴露，原因可能有很多：
a. 数据量积累大了，业务响应能力降低
b. 流量积累大了，现有计算力不够（并不是横向扩容就能解决的）
c. 用户操作花样倍出，防不胜防
d. 依赖的服务逐渐不稳定（包括网络、基础设施等）
所以降级方案是必须要有的，防止出现问题之后没有退路。降级方案可以说是一个长期的及时止损方案，灰度是一个短期的及时止损方案。
降级方案用表格给出业务模块或接口、降级方式（自动、手动）、降级是否对主要业务流程有损失、若有损失修复方案。在设计降级方案时，一定要与产品、测试、业务人员进行充分沟通，说明降级方案，一起讨论可行性。

模块	主流程	是否可降级	降级流程	是否有损（对主流程）	备注

9. 异常处理

有了降级方案依然不能代表你的业务会 100% 可靠。完成功能谁都可以，但是能把所有异常情况都考虑到，是最难的。没有人能考虑到所有异常，但是只要考虑到，总会让你的业务更可靠一点。

异常处理也是需要通过表格，主要写明以下几个阶段：

a. 异常出现阶段：给出异常情况

b. 异常处理阶段：是否可以降级，修复异常方案（精确写出处理流程，有必要的话给出流程图）

c. 异常恢复阶段：给出带来的损失以及数据如何修复

模块	异常情景	处理流程	降级方案	损失与数据恢复

10. 容量预估

容量预估里包含了流量预估，这一部分是可以保证你的业务不会在极端情况下压垮（大数据量、高并发）。

流量预估

用表格的形式给出你的接口平均 QPS、峰值 QPS、接口请求和返回报文大小，消息队列的平均消息数、峰值消息数、报文大小。

这一部分如果是改动的业务，可以参考以前的监控，如果是新业务一定要拉运营、产品确定业务量。若预估峰值会很高，一定要进行压测。

如下表：

模块	接口 / Topic	报文大小	上线一周	上线一个月	上线半年	上线一年

容量预估

列出表格，指定业务模块，给出你用到的所有存储在一段时间的数据量增长：

a. 上线一周：评估灰度时会产生的数据量，有助于推导出后续产生的数据量

b. 上线一个月：此时已经稳定的全量跑了一个月，如果业务比较火爆，这个时候产生的数据量足以计算出半年和一年的数据量

c. 上线半年：如果你的业务并不火爆，半年的数据量基本是定型了。你的业务未来发展也不会增长太多。但是如果你的数据量预估是一个比较大的值，就要考虑方案是不是可以扛得住这个数据量。

d. 上线一年：这个时候随着数据量的积累，可能会暴露出一些问题。需要考虑你的方案在每个细节方面是否可靠了。

这里有一个需要注意的，并不是只是评估存储大小就够了，如果预估你的流量很大，一定要精确计算出你的一个请求过程产生的对象大小，防止应用服务被频繁 gc Hang 死。

模块	表 / 索引	上线一周	上线一个月	上线半年	上线一年

11. 监控报警

上面的一切做完，依然不完美，出现问题要及时的发现，有一个 nb 的监控策略是很重要的。这里要用表格的形式给出关键模块或接口的异常报警形式（短信、邮件、IM、电话等），给出核心的业务指标和非业务指标（QPS、响应耗时、消息积压数等），指标出现异常（低于某个阈值）要进行报警。如果有监控系统可以接入，给出需要接入的指标参数。

如下表：

模块	指标	阈值	报警形式

12. 参考文档

给出产品设计文档、需求文档、以往的方案设计文档、接口文档、依赖基础组件或服务的说明文档。给出文档链接或附件，方便 review 的过程中直接打开看。

总结

1. 写技术方案是个漫长的过程，是个细致活，通常要至少三天，一周甚至一周多的时间。技术方案写的越完善，可靠和可行性越强。
2. 技术方案除了考虑功能性设计、架构，要多关注应急方案，关注可靠性。
3. 可以用大量的图和少量的文字来实现技术方案。
4. 技术方案完成后要找多人进行 review。
5. 业务性较强，拆分微服务，用 DDD 的思想进行设计会很有效果。
6. 如果开发过程中技术方案有变更，一定记得及时修改，维护好这份技术方案，是面试吹 nb 的本钱，也是为后来人做准备。
7. 这里谈的是一个可实施的技术方案，并不是交付给客户和糊弄国家项目的吹 nb 方案，客户一定不会喜欢这样的方案。

> 2021 年 03 月 28 日记：有很多人私信我要方案模板，写了一个新的文章，需要可自取：https://blog.csdn.net/kid_2412/article/details/115282680