大型分布式网站电商项目-慧科商城

1 项目计划

1.1、课程前言

1、必须具有一定的开发基础，CRUD 不是我们的菜
2、注重的是实际业务场景
3、注重的是问题的解决方案
4、注重的是架构的思路

1.2、课程特色

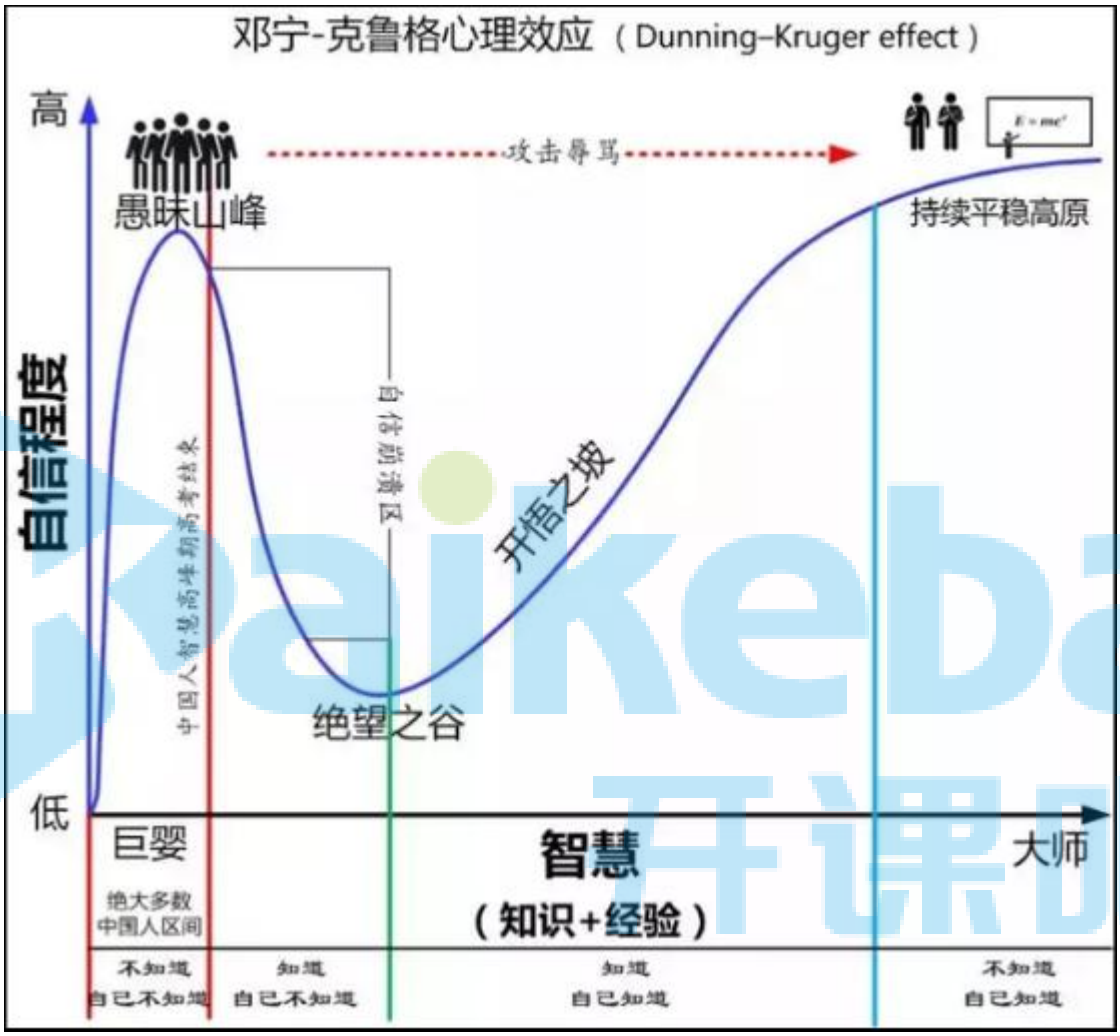
- 提升架构高度，仅仅寄希望于代码层级是远远不够的。
  - 代码解决的的执行力的问题 ，架构更多的是依赖 业务的洞察能力 和 技术视野
- 课程重点
  - 架构解决方案
    - 技术解决方案落地
  - 架构背后思考
  - 核心问题解决方案

1.3、课程问题

- 项目实战 和 其他的 课程到底有什么区别？
- 课程中是否敲代码？（源码发给大家，CRUD 不写）

2 架构师认知

2.1、架构师成长之路



2.2、架构是什么？

1）对业务场景抽象后得出的支持骨架

老板：100w 日活量，10W QPS 微服务架构

2）架构为业务场景而生、被业务场景而弃

老板：10 天上线

3）架构没有最好、只有“最合适”（人员技术研发能力、业务复杂度、数据规模大小、时间成本、运维能力….）

4） “最合适”架构都是业务场景折中（Balance）的选择

总结：选择架构时候，必须选择最适合公司当下环境的架构。

2.3、架构目标是什么？

用户网站访问调查：reponse time : 3s ---- 60% 用户流失

RT 时间：ms 高性能 （前端：美观大气的上档次页面—非常简单，后端：一系列的优化 ms）

高可用： 任何时候项目都必须可用

可升缩： 大促，流量瞬间增大....

可扩展： 开发角度（新需求进行迭代），扩展

安全性： 必须

敏捷开发： 可持续交付，可持续部署

架构师目标： 问题： 采用什么样方式，才能构建以上目标的项目？？

2.4、架构模式？

单体： 10 台
微服务： 10 服务 x 10 = 100 台

2.5、高性能架构

以用户为中心，提供快速的网页访问体验。主要参数有较短的响应时间、较大的并发处理能力、较高的吞吐量与稳定的性能参数。

可分为前端优化、应用层优化、代码层优化与存储层优化。

- 前端优化：网站业务逻辑之前的部分；--- vue ,react +nodejs – 工程化
- 浏览器优化：减少 HTTP 请求数，使用浏览器缓存，启用压缩，CSS JS 位置，JS 异步，减少 Cookie 传输；CDN 加速，反向代理；
- 应用层优化：处理网站业务的服务器。使用缓存，异步，集群，架构
- 代码优化：合理的架构，多线程，资源复用（对象池，线程池等），良好的数据结构，JVM 调优，单例，Cache 等；
- 存储优化：缓存、固态硬盘、光纤传输、优化读写、磁盘冗余、分布式存储（HDFS）、NoSQL 等

2.6、高可用架构

大型网站应该在任何时候都可以正常访问，正常提供对外服务。因为大型网站的复杂性，分

布式，廉价服务器，开源数据库，操作系统等特点，要保证高可用是很困难的，也就是说网站的故障是不可避免的。

业务上也需要保证，网站高可用性。（bug,异常）

例如：

对输入有提示，数据有检查，防止数据异常。

系统健壮性强，应该能处理系统运行过程中出现的各种异常情况，如：人为操作错误、输入非法数据、硬件设备失败等，系统应该能正确的处理，恰当的回避。

因软件系统的失效而造成不能完成业务的概率要小于 5‰。

要求系统 7x24 小时运行，全年持续运行故障停运时间累计不能超过 10 小时。

系统缺陷率每 1,000 小时最多发生 1 次故障。

在 1,000,000 次交易中，最多出现 1 次需要重新启动系统的情况。

如何提高可用性，就是需要迫切解决的问题。首先，需要从架构级别考虑，在规划的时候，就考虑可用性。
不同层级使用的策略不同，一般采用冗余备份和失效转移解决高可用问题。

- 应用层：一般设计为无状态的，对于每次请求，使用哪一台服务器处理是没有影响的。一般使用负载均衡技术（需要解决 Session 同步问题）实现高可用。

- 服务层：负载均衡，分级管理，快速失败（超时设置），异步调用，服务降级，幂等设计等。

- 数据层：冗余备份（冷，热备[同步，异步]，温备），失效转移（确认，转移，恢复）。数据高可用方面著名的理论基础是 CAP 理论（持久性，可用性，数据一致性[强一致，用户一致，最终一致]）

总结：
1、集群
2、数据冗余
3、（网络抖动：请求超时，请求失败）---- 重试（成功）--- 可用的
4、验证（防止服务漏洞）
5、极端异常情况，必须回避 ---- （每一行代码都有可能出现异常：数据一致性）
6、缓存
7、异步

2.7、可伸缩架构

伸缩性是指在不改变原有架构设计的基础上，通过添加/减少硬件（服务器）的方式，提高/降低系统的处理能力。

- 应用层：对应用进行垂直或水平切分。然后针对单一功能进行负载均衡（DNS、HTTP[反向代理]、IP、链路层）。
- 服务层：与应用层类似；
- 数据层：分库、分表、NoSQL 等；常用算法 Hash，一致性 Hash

云原生：项目运行云端，可以随时动态扩容—K8S

8 核心+16G : 2000QPS +- (此数字是估算结果，真实结果受到代码编写数据结构，业务逻辑，架构、rt,以现实测试结果)

2.8、可扩展架构

SOA,微服务 --- 根据业务拆分模块 ----- 新业务需求 ---- 根据新的业务需求创建一个新模块服务

可以方便地进行功能模块的新增/移除，提供代码/模块级别良好的可扩展性。

- 模块化，组件化：高内聚，低耦合，提高复用性，扩展性。
- 稳定接口：定义稳定的接口，在接口不变的情况下，内部结构可以“随意”变化。
- 设计模式：应用面向对象思想，原则，使用设计模式，进行代码层面的设计。
- 消息队列：模块化的系统，通过消息队列进行交互，使模块之间的依赖解耦。
- 分布式服务：公用模块服务化，提供其他系统使用，提高可重用性，扩展性。

2.9、安全架构

对已知问题有有效的解决方案，对未知/潜在问题建立发现和防御机制。对于安全问题，首先要提高安全意识，建立一个安全的有效机制，从政策层面，组织层面进行保障，比如服务器密码不能泄露，密码每月更新，每周安全扫描等。以制度化的方式，加强安全体系的建设。同时，需要注意与安全有关的各个环节。安全问题不容忽视，包括基础设施安全，应用系统

安全，数据保密安全等。

- 基础设施安全：硬件采购，操作系统，网络环境方面的安全。一般采用正规渠道购买高质量的产品，选择安全的操作系统，及时修补漏洞，安装杀毒软件防火墙。防范病毒，后门。设置防火墙策略，建立 DDOS 防御系统，使用攻击检测系统，进行子网隔离等手段。

- 应用系统安全：在程序开发时，对已知常用问题，使用正确的方式，在代码层面解决掉。防止跨站脚本攻击（XSS），注入攻击，跨站请求伪造（CSRF），错误信息，HTML 注释，文件上传，路径遍历等。还可以使用 Web 应用防火墙（比如：ModSecurity），进行安全漏洞扫描等措施，加强应用级别的安全。

- 数据保密安全：存储安全（存储在可靠的设备，实时，定时备份），保存安全（重要的信息加密保存，选择合适的人员复杂保存和检测等），传输安全（防止数据窃取和数据篡改）；

常用的加解密算法（单项散列加密[MD5、SHA]，对称加密[DES、3DES、RC]），非对称加密[RSA]等。

2.10 敏捷开发

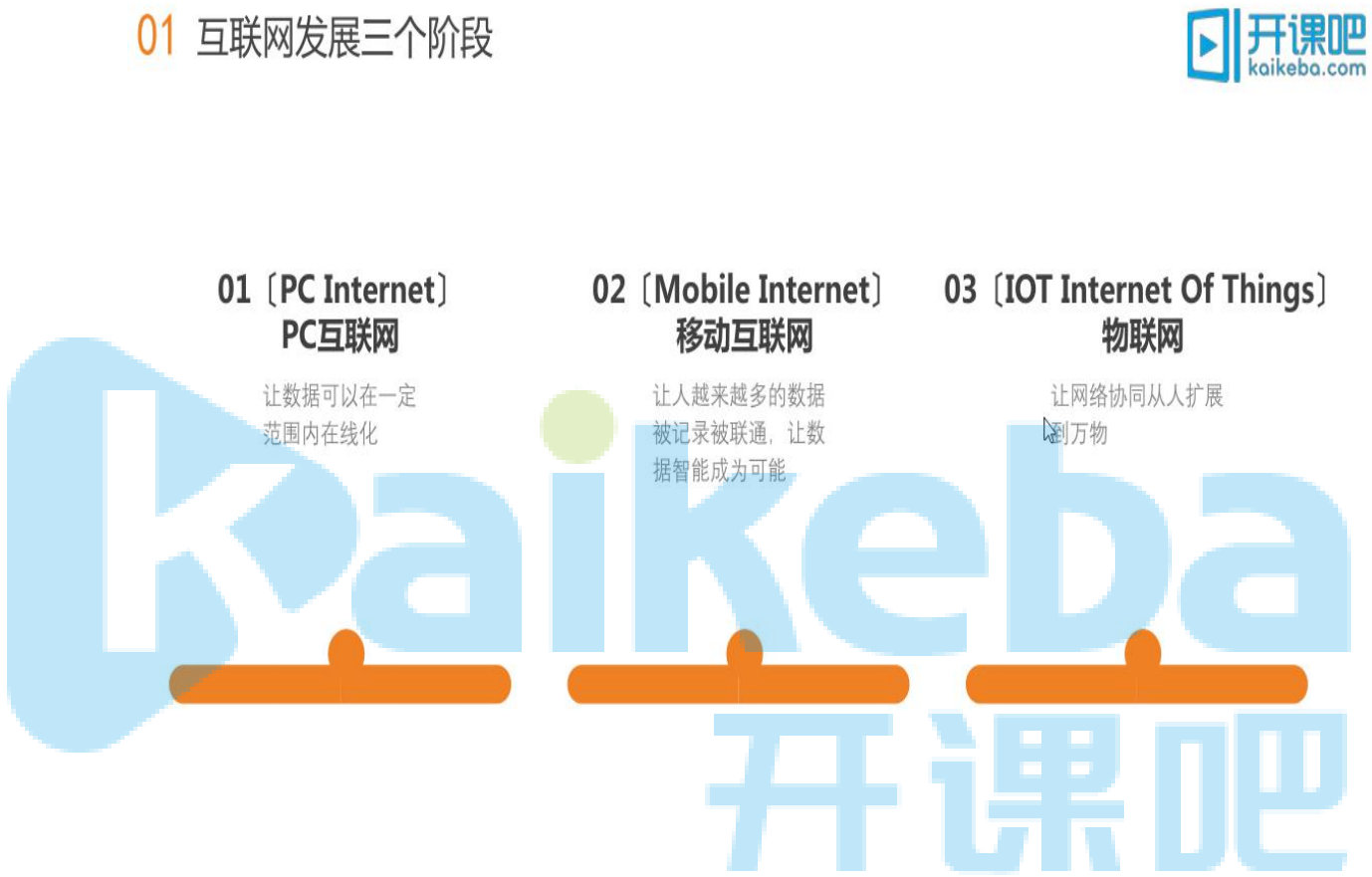
敏捷开发 --- 可持续交付 ，可持续部署 --- 微服务架构

Jenkins + maven + git + docker + k8s ========== 流水线生产模式 ======== 降本增效

5% 运维减少：200% 工作量

3 互联网认识

3.1 互联网发展



3.2 Web 演进

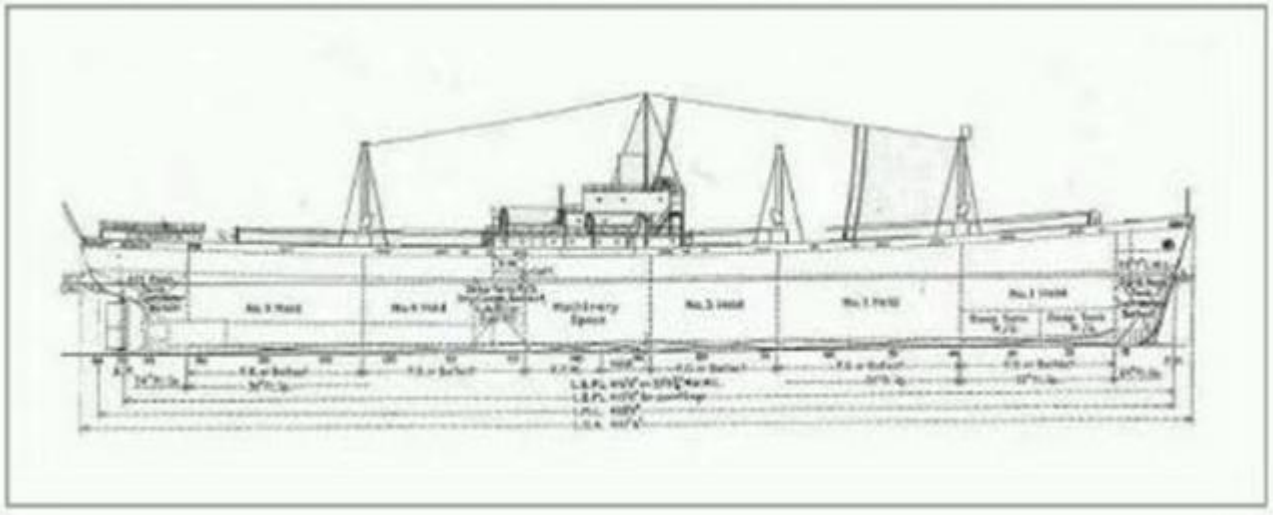
3.3 发展特点

3.4 上线全过程

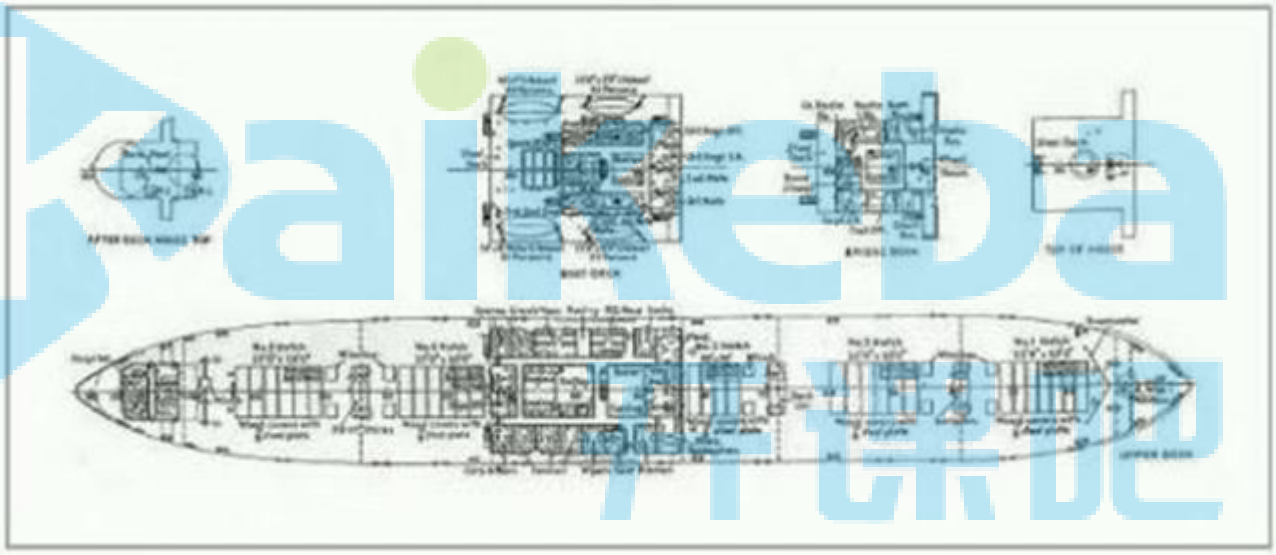
客户（老板）需求：造一条船，能过河就好

产品经理：我们可以提供这样的方案

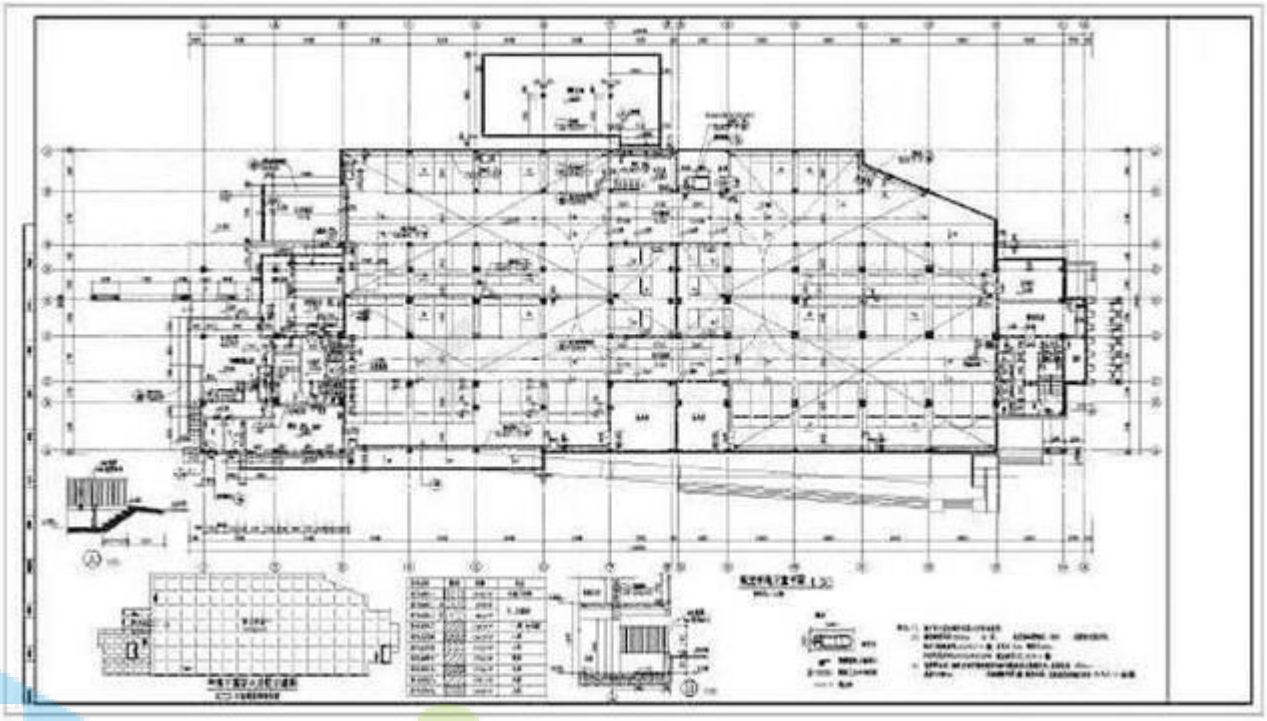
首席架构师：按照需求规划蓝图



高级研发经理：进行项目分解



技术平估：这个项目至少需要 1.5 年



老板发话：市场不等人，先上线再迭代，给技术团队 1 个月时间！

研发团队：重新更改设计，马上开始编码

测试团队：提前进入单元测试阶段

测试团队：突进入集成测试阶段

项目终于止式上线了

船动了，产品实际跑起来了

可伶了这帮苦逼的人肉运维

市场部：升始对外宣传成功案例

4 大型分布式网站架构演进思考

4.1、架构演进

单体架构 ----> 水平拆分 / SOA 架构 ----> 微服务架构 ----- 项目架构骨架

云原生时代：

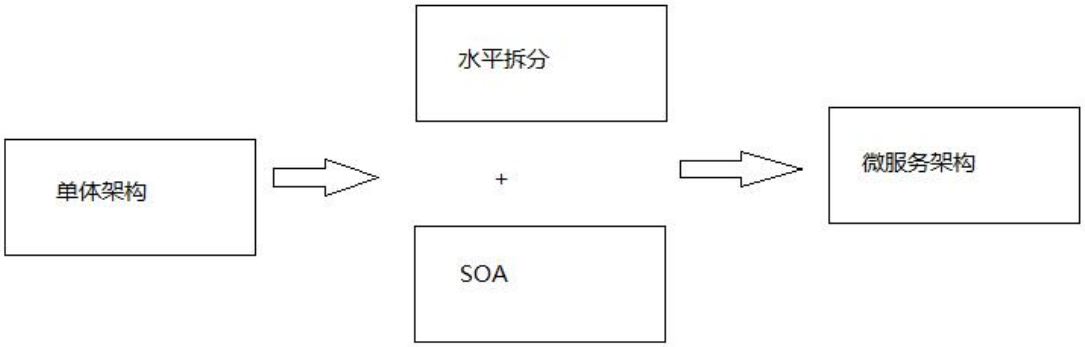
1、service mesh

2、serverless

3、iaas

4、paas

5、saas



架构为什么从单体架构演进到微服务架构？？

1、提高项目并发能力

2、CI/CD

3、可伸缩，可扩展

4.2、单体架构

单体架构： 您认为单机并发量有多少呢？？ 是否可以做出一个估算呢？？

服务器： 2cpus 4GB ---------- 对这个服务并发能力进行估算？？？

前提条件： 忽略其他因素影响（reponse time，cpu 切换时间），只需要根据内存估算服务器并发能力。

并发： 200 QPS

静态请求： 150 ---- 访问数据在缓存，不需要读数据库，也就是不需要把数据放入内容 ------ 2M

动态请求： 50 ---- 查询数据库，把数据放入内存 -------- 10M

计算：

总内存：

150* 2 + 50 * 10 = 300M + 500M = 800M

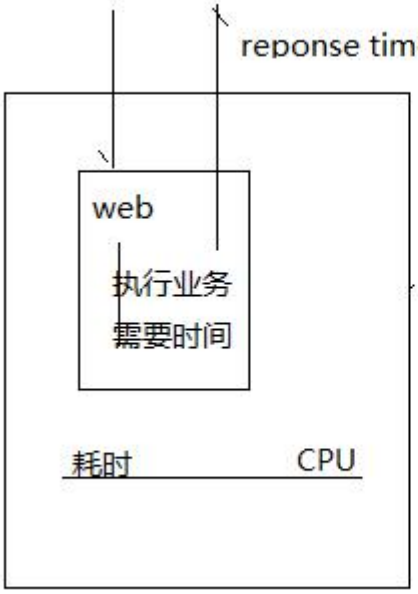
思考： 服务器总内存一共有4G ,操作系统占用一部分内存，剩下的内存才是服务在使用？？

800M * 4 = 3.2 G （内存） ---- 200 * 4 = 800 QPS

800M * 5 = 4G （内存） ---- 200 * 5 = 1000 QPS

2cpus ,4GB --- QPS (0,800) ---- 估算值（只考虑内存因数） ——————————↘ 300+-

4cpus,16GB --- QPS (0,3200) ---- 估算值（只考虑内存因数） ——————————↘ 2000+-



并发能力影响因数：

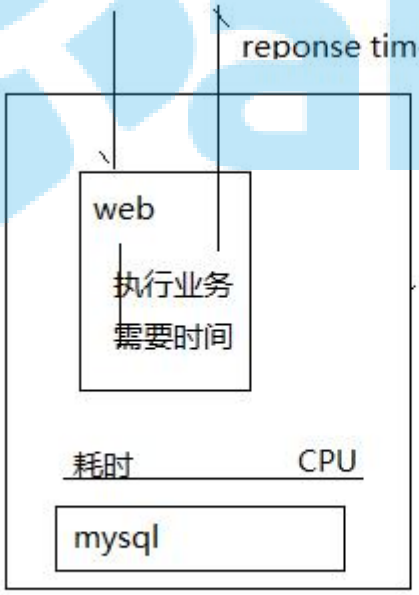
1、rt

2、cpu切换时间

3、内存

真实结果：以实际测试结果为准。

单体架构： 服务没有进行拆分。



并发能力影响因数

1、rt

2、cpu切换时间

3、内存

优点：

1、部署简单

2、测试简单

3、开发简单

4、集群简单

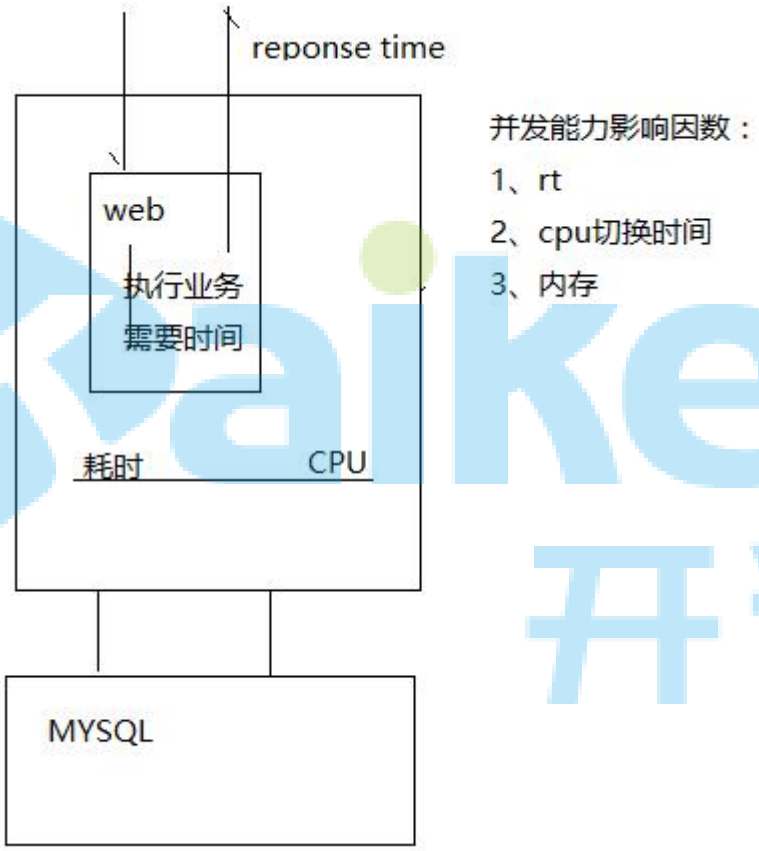
缺点：

1、无法大规模扩展集群（无法适应海量并发项目）

2、无法进行可扩展开发

无可替代的优点：
Rt(reponse time): 响应链路非常短，导致响应的时间非常短。
1、对于一些对数据实时性要求较高的项目，或者要求快速响应的项目，必须采用单体架构。（股票）
2、创业型公司， 采用单体架构也是 ok 的。

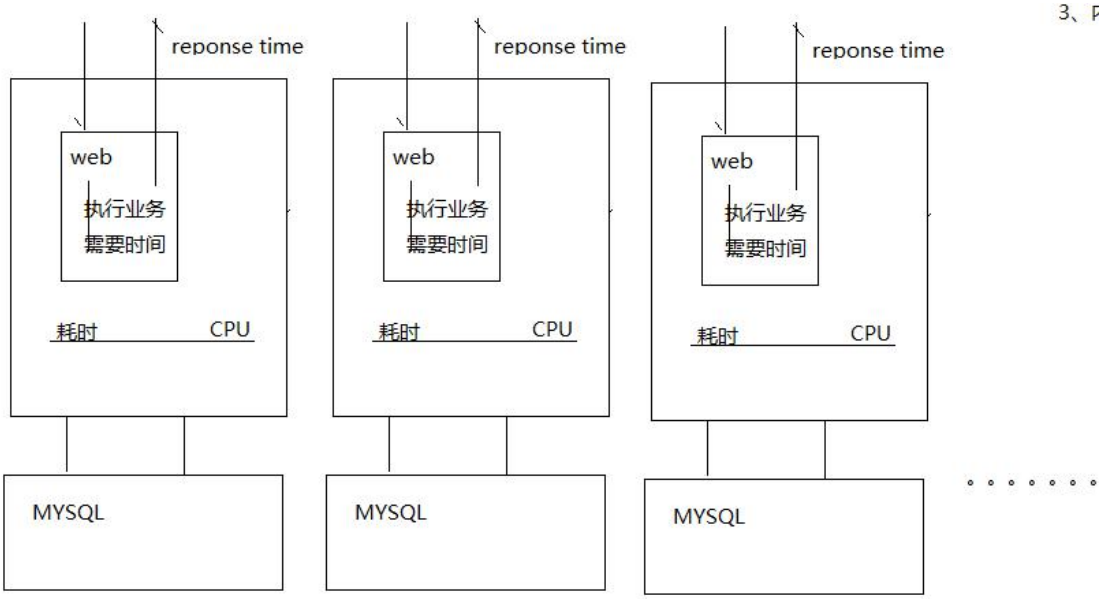
单体架构优化:
1）数据库 * 项目进行分离部署



Cpu,内存，io 相对来说，就可以应用更多的并发请求。

2）单体架构优化 --- 集群

3、P



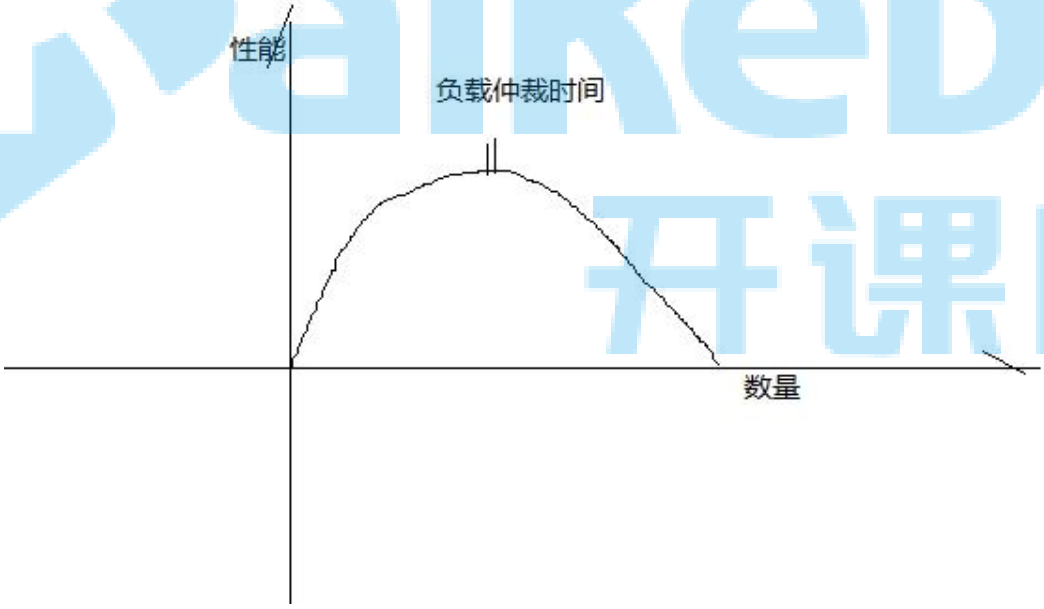
集群部署： 进一步提高网站并发能力（吞吐量），因此使用集群可以应对大部分网站需求。

思考：

网站流量越来越大，如何进一步提升网络的并发能力？？？

解决方案：

无限制扩展服务器集群规模



还需要考虑其他因素：

1、成本

2、是否可行

4） 缓存

使用 memcached , Redis,ehcache(堆内存，堆外内存) 构建服务缓存。

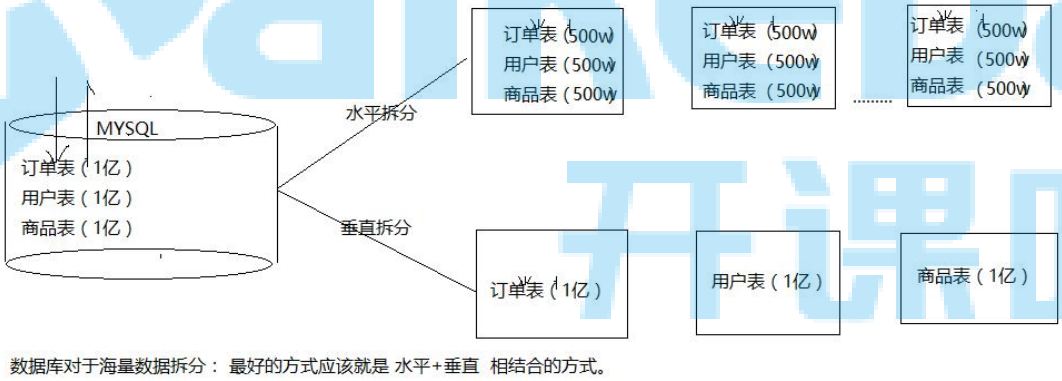
降低：rt, 提高并发能力。

5）数据库优化
* 分表，分库
* SQL 语句优化
* 表设计优化
* 开启缓存
* 开启索引

6）CDN 缓存，nginx 缓存，浏览器缓存
使用以上缓存措施，减少请求流入下游服务。减轻下游服务的压力，同时提供项目并发能力。

4.3、水平拆分架构

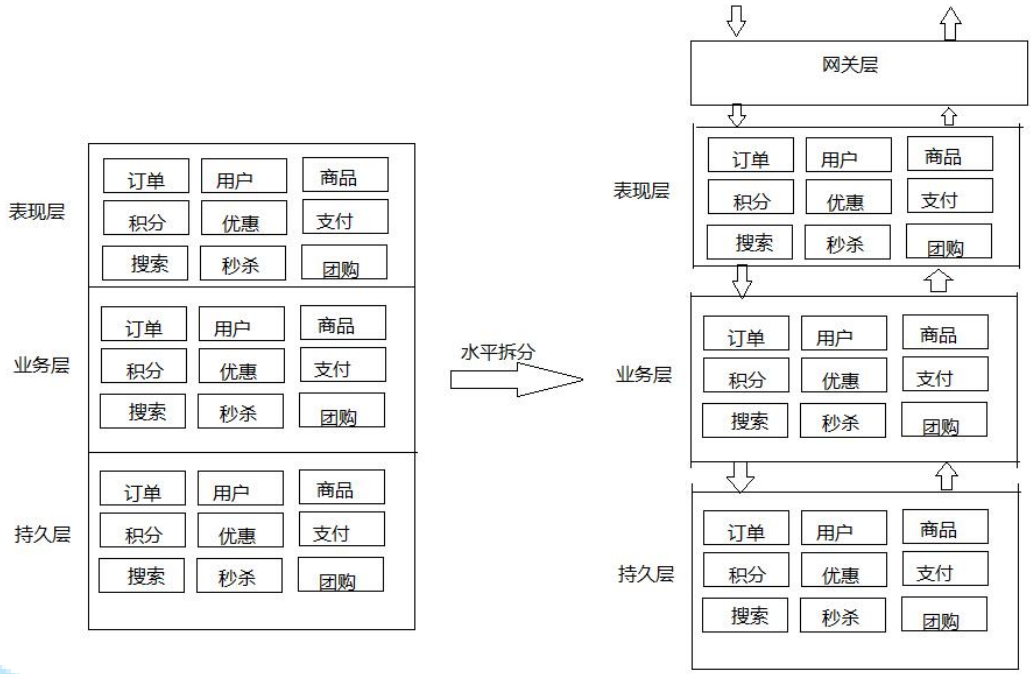
数据库： 海量数据需要存储，需要对数据库进行分表分库进行存储。
1、水平拆分
2、垂直拆分
案例： 数据库海量数据数据库拆分方式



数据库对于海量数据拆分： 最好的方式应该就是 水平+垂直 相结合的方式。

从数据库的拆分方式，能否去思考项目的架构方式：
1、水平拆分 ---- 对业务进行分层拆分方式
2、垂直拆分 ---- 对业务进行拆分。

什么是水平拆分架构：



水平拆分架构问题：

1、请求链路变长

2、reponse time 变长

3、定位问题变得困难

4、运维成本上升

5、拆分粒度比较粗

6、模块耦合度高

分析问题： 水平拆分 rt 变长，是否意味着 QPS（吞吐量） 变小了？？？

分析原因：

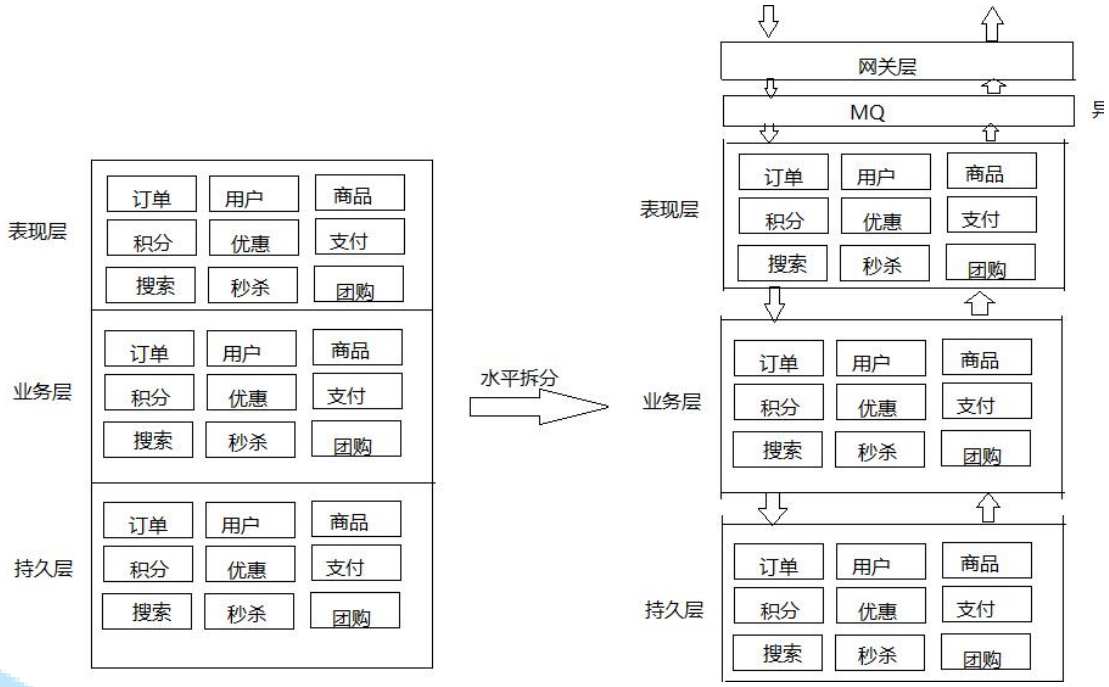
1、虽然链路变长了，但是服务进行分层拆分，每一层代码执行变少了，且分得了更多 CPU，内存资源，处理速度更快。

2、水平拆分：集群扩展规模变得更加庞大

水平拆分： 吞吐能力是提升的。

问题：是否还有解决方案，提升水平架构拆分的吞吐能力？？

解决方案： 异步架构



在网关层和服务层之间加入 mq 消息中间件，让请求到达服务之间，进行异步化的处理。从而可以进一步提高吞吐能力。

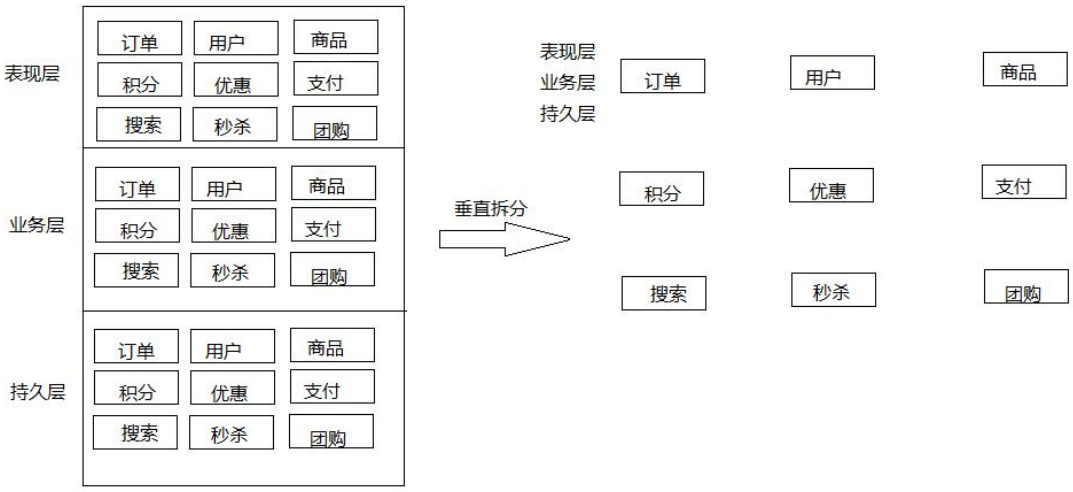
思考问题：

业务操作，读，写操作，那个操作适合异步架构，那个操作不适合异步架构？？

解决方案： 读操作不适合异步架构

写操作适合异步架构

4.4、垂直拆分



垂直拆分： SOA 架构，根据业务进行拆分，把不同的业务拆分为单独的模块。
SOA 架构有什么问题？？
1、每一个服务还是一个单体架构
2、对总线依赖较高

优点：
1、吞吐能力急剧提升
2、敏捷开发
3、可扩展性

4.5、微服务架构

水平拆分 + 垂直拆分 == 微服务架构。

4.6、架构思考

1、单体架构
2、水平拆分
3、SOA 架构
4、微服务架构
云原生时代：
1、iaas
2、paas
3、saas
4、云原生架构(服务治理，服务监控，部署，运维)
5、serverless

思考： 企业需要什么样架构？
1、老板需求？
2、人员技术能力？
3、时间成本？
4、运维成本？
5、业务复杂度
6、架构复杂
例如： 采用微服务架构？？
思考问题：
1、服务治理
2、数据一致性
3、接口幂等性

总结： 选择适合当前公司架构，就是最好的架构。