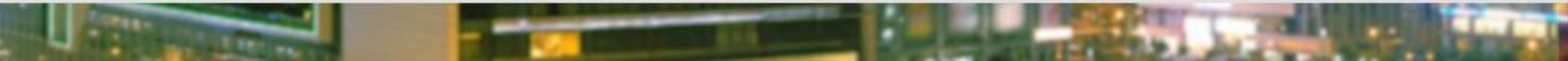
中小企业架构设计之道

个人介绍

柳鋆

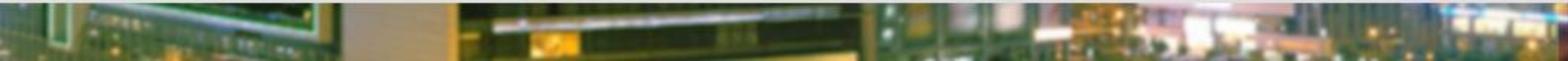
前美味七七架构总监

dl2k@163.com



为什么要谈中小企业的架构之道

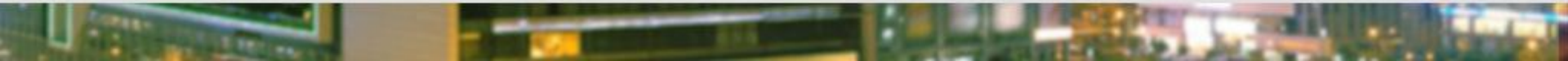
- 所有的大公司都是从中小型企业发展起来的
- 如今的创业环境下中小企业是大多数
- 中小企业的IT现状堪忧
- 当业务变化时打了太多的补丁程序
- 当业务发展时系统不断出现各种瓶颈
  - 系统出现各种故障、卡顿
  - 数据库经常锁死
  - 用户反应网站打不开
  - 业务团队或者老板抱怨一搞活动系统就挂



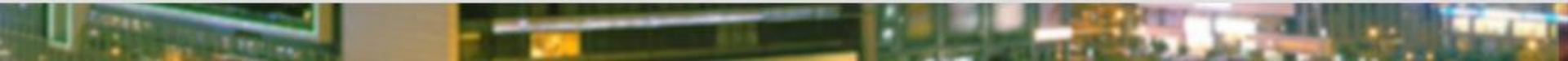
谈谈中小企业的现状

- 技术团队人数不超过50人 [100W薪资包]
- 硬件数量20-30台服务器 [不同批次、不同品牌]
- 带宽出口100Mbits [大概也就10MByte每秒]
- 项目耦合度高[一个站点涵盖了几乎大多数功能]
- 数据库集中[可能没有主从备份、没有读写分离]

- 网上看到的BAT的技术架构根本无法简单复制
  - 部署成本和时间严重不足
  - 现有代码重构量大
  - 技术人员承载力不足



我们应该如何面对这些？



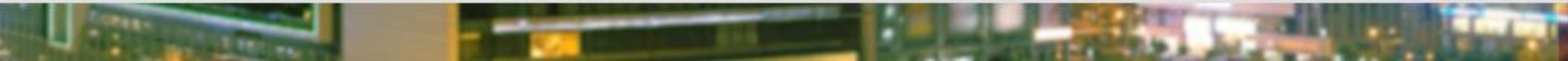
我们的武器库有那些武器 - 常规武器

- 数据库
  - 主从复制
  - 读写分离
  - 分库分表
  - KV数据库
  - 内存数据库
- 运维层面
  - 静态资源分离
  - 外部图片缓存
  - Nginx反向代理
  - Docker容器化
  - 引入云的伸缩方案
  - 负载均衡器
- 技术改造
  - 最大层面使用缓存

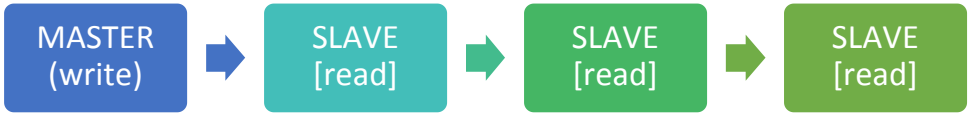
内存数据库

静态资源分离

消息机制与解耦



- Redis 特别推荐
  - 稳定 – 只要没有内存用满，长时间稳定运行
  - 性能出色 – 单节点轻松承载每秒数千次访问
  - 结构多样 – 除了KV存储外，队列、集合等
  - 集群 – 在3.0以后支持集群，容量不再受到局限
  - 安全 – 支持主从、包括一写多读的链条状



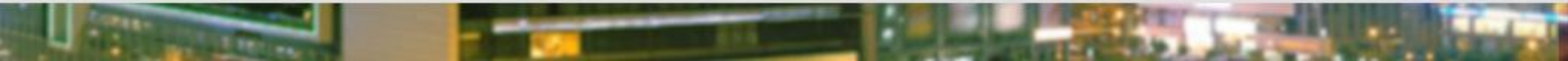
- Memcached
  - 作为缓存性能出色
  - 有数据量限制
  - 自动淘汰数据

再来说说静态资源分离

- 故事 or 事故
  - 有了微信公众号，一推送活动网页就打不开了
  - 发现我们的出口140Mbps的带宽全满了
  - 我们把活动页面里面的图片和js都放到CDN上
  - 单量迅速恢复、后续每周活动都在提升

- 后续改进
  - 使用七牛来存放商品及活动的图片及js文件
  - 开发了专门的服务控制所有前后台图片上传直接到cdn上

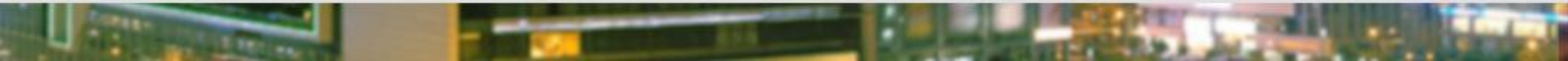
- 静态资源分离的本质是流量成本的精细化管理
  - 带宽成本
  - 业务站点服务器的计算资源成本
  - 资源存储的成本



最后讲讲消息机制与解耦

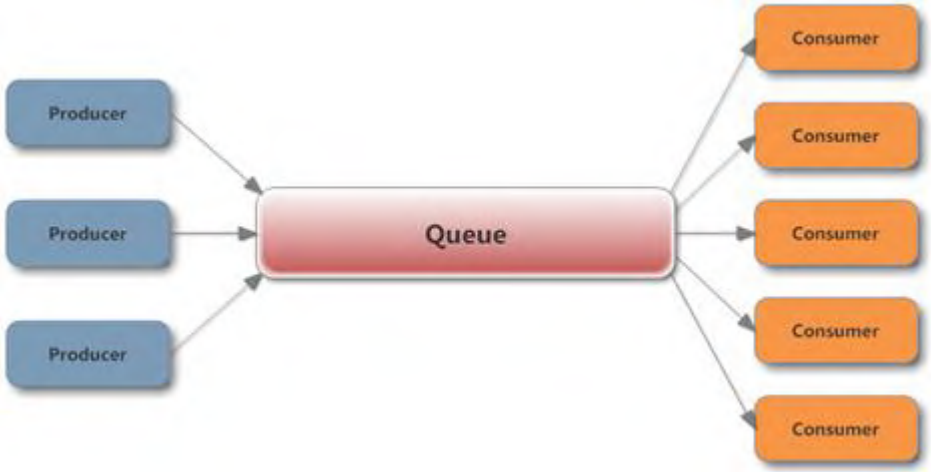
- 中小企业面对主要挑战
  - 大访问量 – 峰谷比高、成本压力大
  - 业务复杂度 – 模式太多，请求耗时长
  - 故障排查与监控

- 解决方向
  - 伸缩性要强
  - 逻辑解耦



回忆下生产者消费者模式

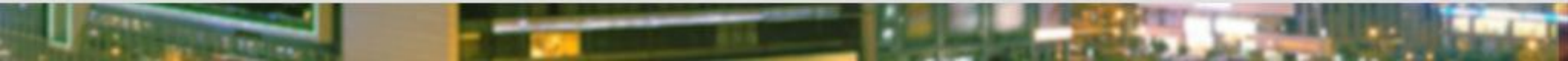
- 三个要素
  - 生产者[Producer]
  - 队列[Queue]
  - 消费者[Consumer]



- 在架构中使用
  - 产生事件的服务[Event]
  - 消息队列[MessageQueue]
  - 处理消息的服务[Handler]

什么是消息机制

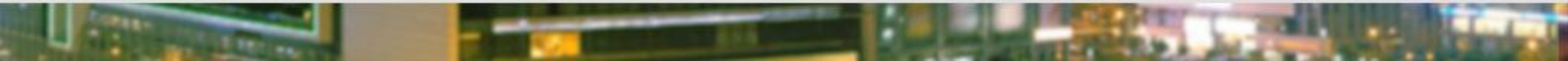
- 定义消息
  - 消息ID - 不重复
  - 消息名称 - 区分不同消息
  - 消息体 - 数据
- 订阅者
  - 不同订阅者
  - 订阅者的副本数量
- 生产者
  - 同一个语义只发一个消息
  - 消息数据完整



- **MSMQ** 微软
- **RabbitMQ** Erlang
- **ActiveMQ** java主流
- **RocketMQ** TAOBAO开源
- **MQS** 阿里云
- **SQS** aws

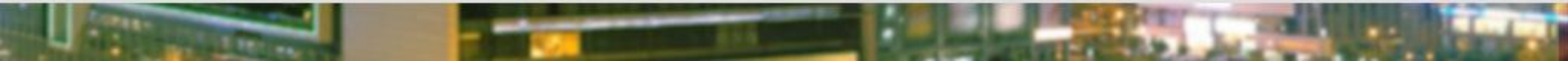
如果你不希望那么重 你可以基于redis自己写一个

- 消息用KV存储
- 消息ID放在队列里



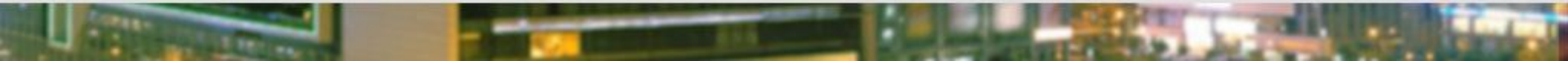
增加订阅者副本来伸缩

- 我们说到了提高系统的伸缩性
  - 当系统压力增加的时候，我们可以轻松通过增加订阅者的副本数量来增加处理能力
  - 善用云服务来提供处理能力，做到快速伸缩
  - 有条件可以使用docker来减少部署的成本
  - 对于可以降级的服务，运用Queue的存储能力来缓冲



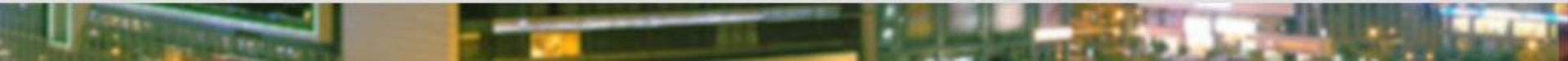
服务拆分的无状态的重要性

- 意义和优点
  - 没有上下文依赖，每个消息都是独立的
  - 尽量不依赖本地配置或者资源
  - 顺序无关性
  - 可随时redo

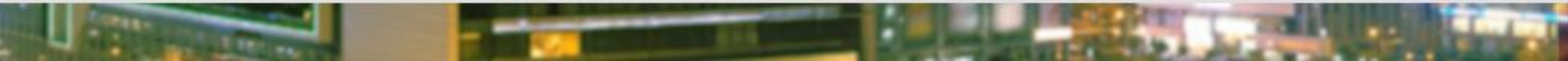


应对必不可少的公共资源

- 线程安全
- 数据库访问，要考虑性能
- 避免各种硬件、系统资源（磁盘）

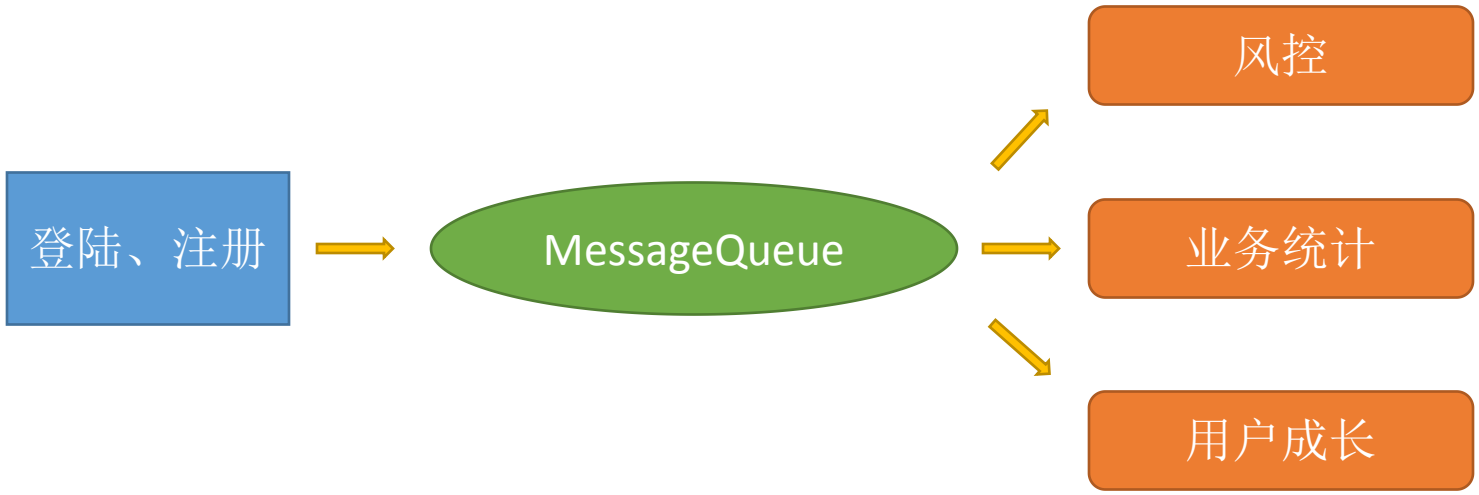


我们来看看有那些应用场景



场景一

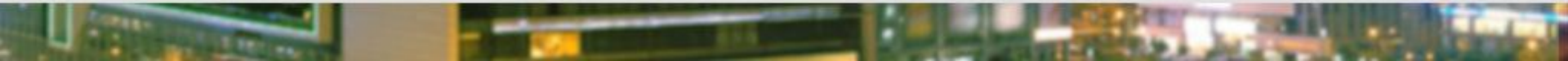
- 业务需求
  - 登陆
    - 风控黑名单
    - 业务数据统计
    - 用户成长体系



场景二

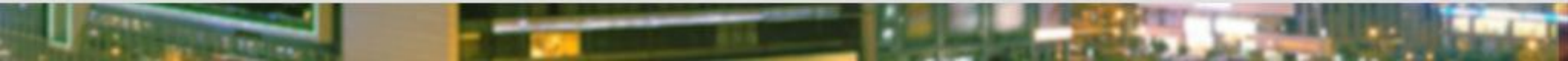
- 业务需求
  - 支付成功，接到银行callback
    - UPDATE payment status
    - CALLBACK bizsystem
    - Statistics

- 在支付场景很常见
- 其实与外部系统的一些状态同步中都很常用
- 还可以跟schedule服务一起联动

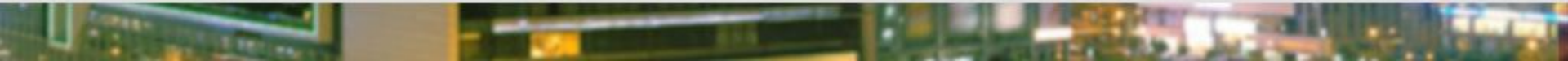


场景三

- 业务需求
  - 数据库更新
    - UPDATE cache
    - DUPLICATE to remote storage
- 我们用这个做了数据库异地复制

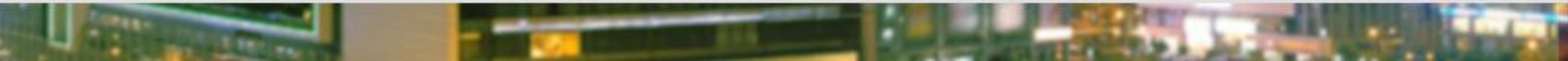


还有那些需要注意的

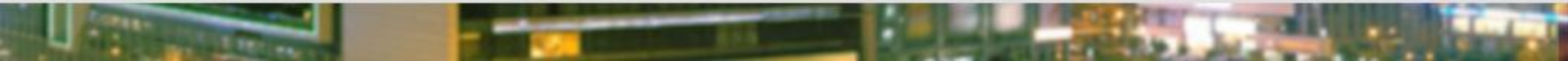


注意事项

- 通常消息是无序的，有序可以做成本高
- 对于重要的消息要考虑MQ是否支持持久化
- 如果所有的订阅者都挂了MQ的存储压力
- 提防丢消息和重复消息
- 此模式下在SCALE时考虑运维的成本



分享一点架构观念



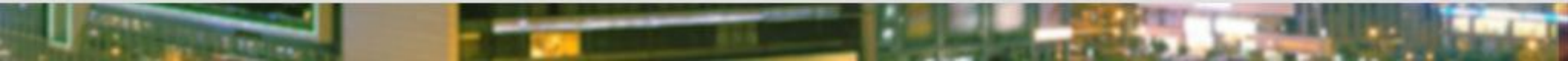
- 避免过度设计
- 始终保持精简，精简才能灵活
- 知识积累很重要，但是首先要解决问题

好吧，重要的事情讲三遍

避免过度设计！

避免过度设计！！

避免过度设计！！！




谢 谢 大 家