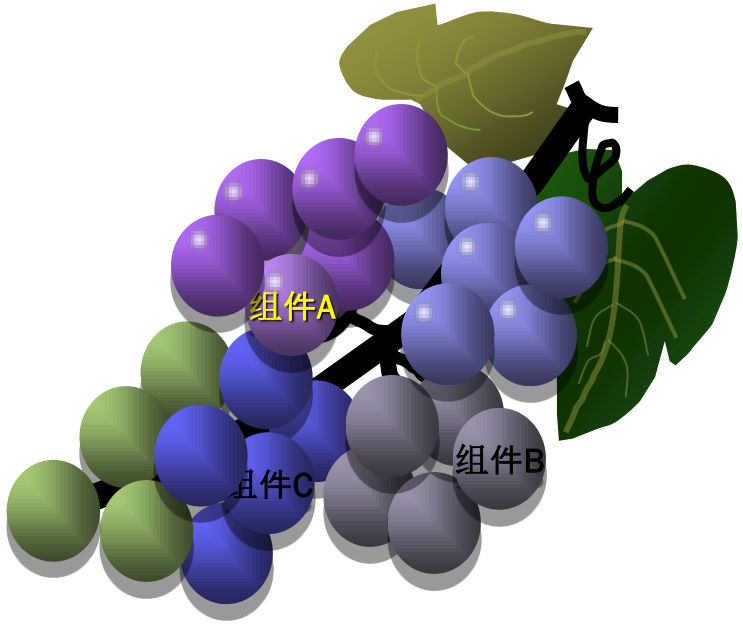
高可用架构-架构知识原理和总体架构

洪光宝 2016年9月

平安金融科技

目录

架构定义

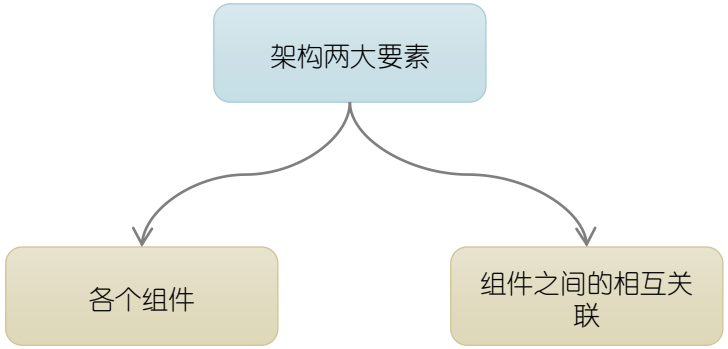


搭积木 → 建房子 → 系统架构

问题1：什么是软件架构

架构定义

- 软件架构是一个系统的蓝图草图，是一种设计方案，类似于建筑物架构概念
- 以作为满足不同客户需求的实际系统设计方案的基础
- 在架构设计原则基础上，对系统的各个部分组合，形成架构

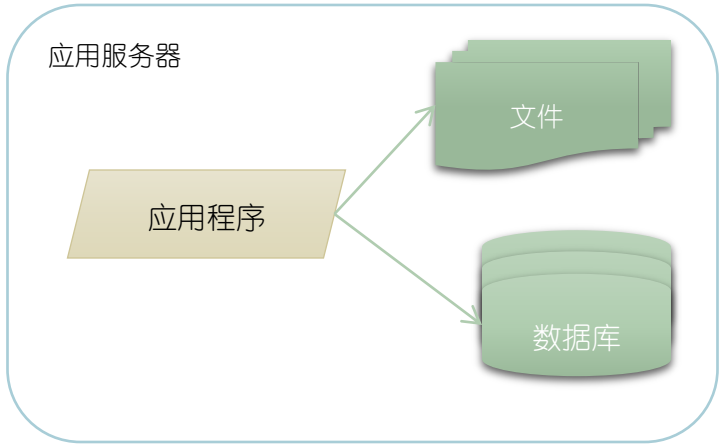


架构定义

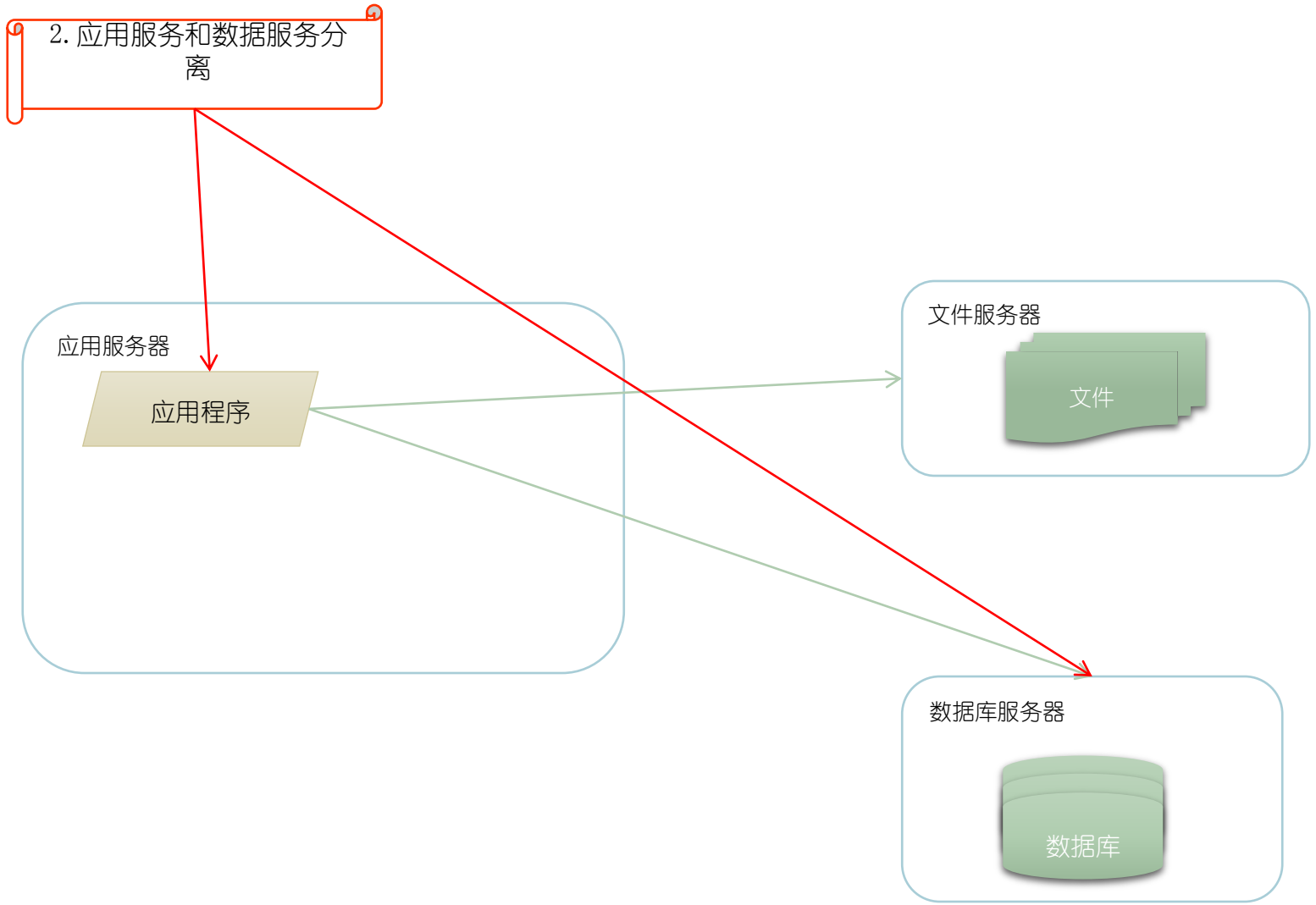
好的架构	坏的架构
◘ 成本最低 ◘ 满足用户需求(将来的功能需求) ◘ 系统稳定性好 ◘ 架构足够灵活 数据量维度 并发量维度 部署运维维度	◘ 成本较高 ◘ 系统复杂、笨重 ◘ 系统不稳定 经常发生故障 ◘ 系统扩展性差 ◘ 系统维护性差 维护代价高

架构演化

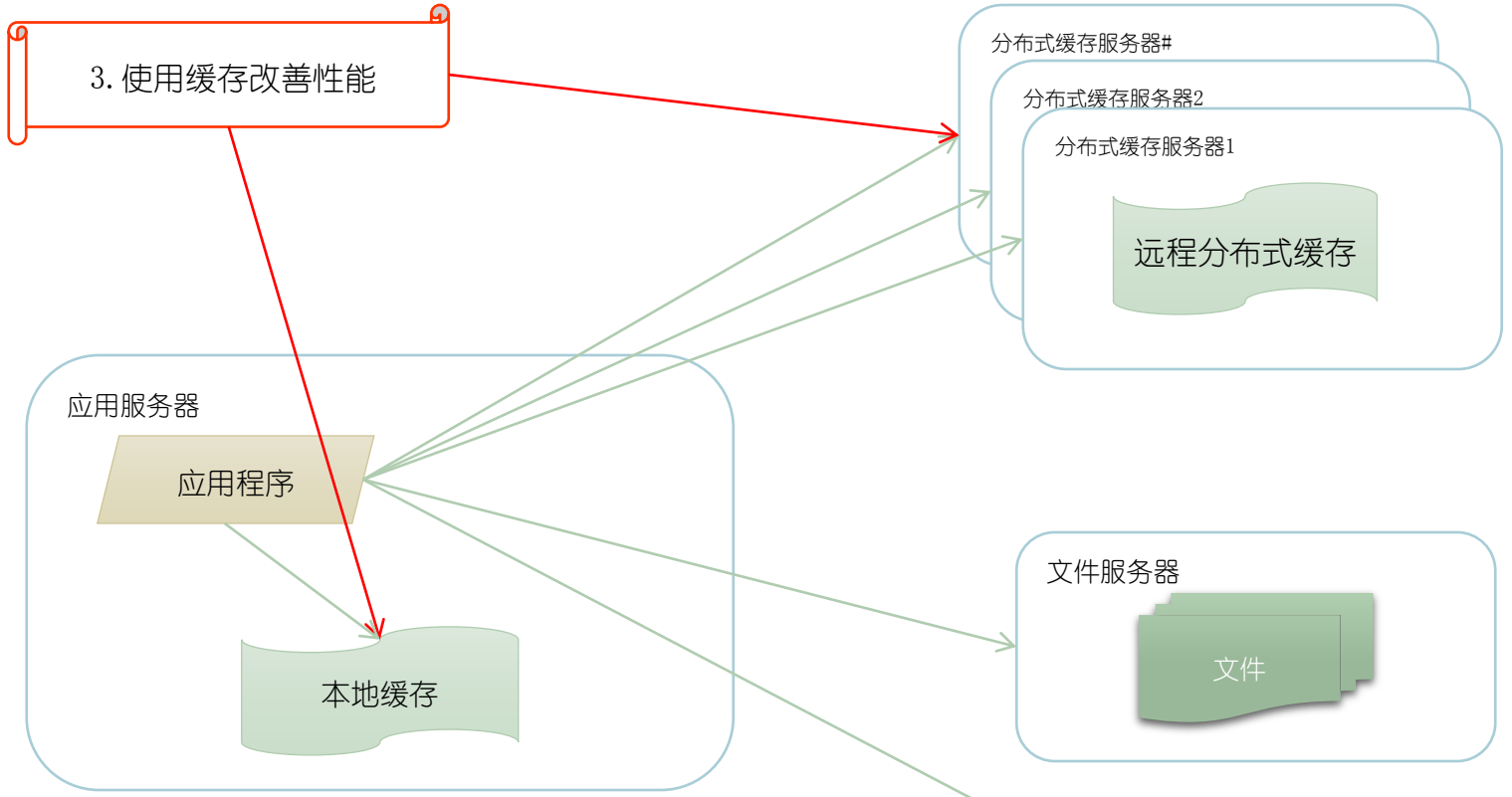
1. 初始阶段的网站架构



架构演化

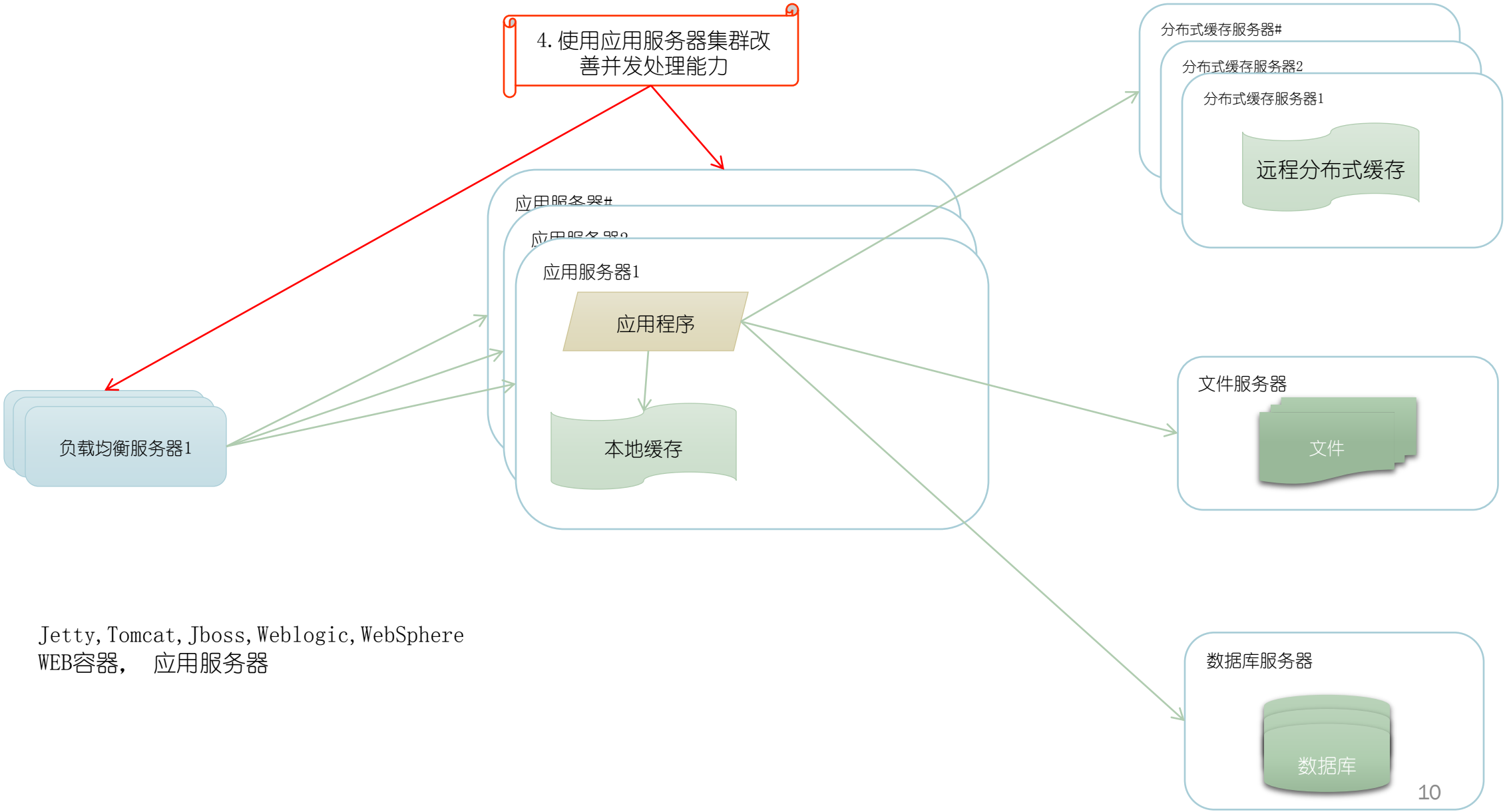


架构演化



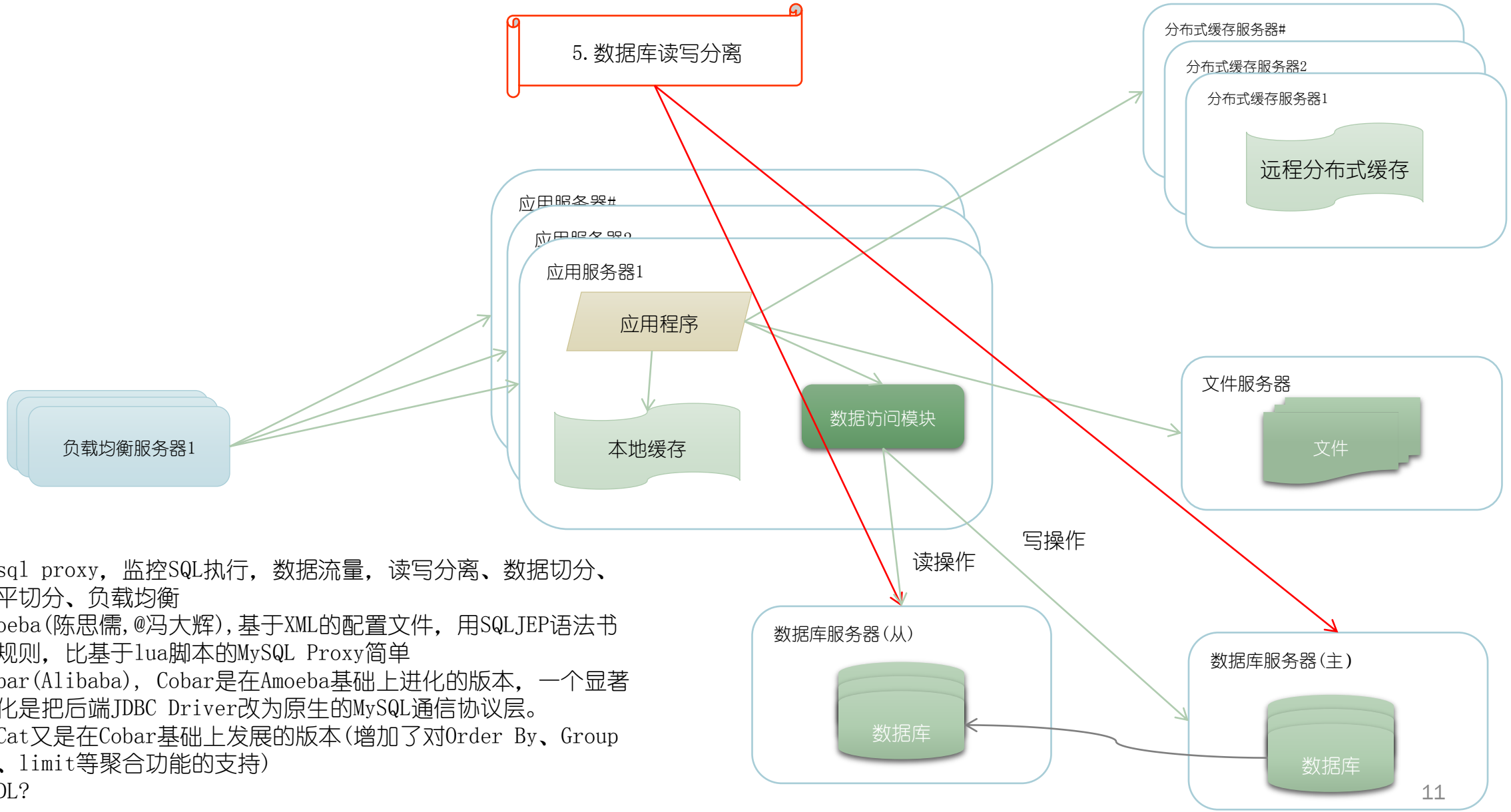
1.进程内缓存
　　自己构造单例、guava、ehcache(heap)
2.本地缓存
　　ehcache(OffHeapStore)被称为BigMemory，只在企业版本的Ehcache中提供，原理是利用nio的DirectByteBuffer实现，比存储到磁盘上快，而且完全不受GC的影响，DirectByteBuffer是通过底层的JNI向C Runtime通过malloc分配，在JVM的GC所管理的堆之外。
3.分布式缓存
　　redis,memcache

架构演化



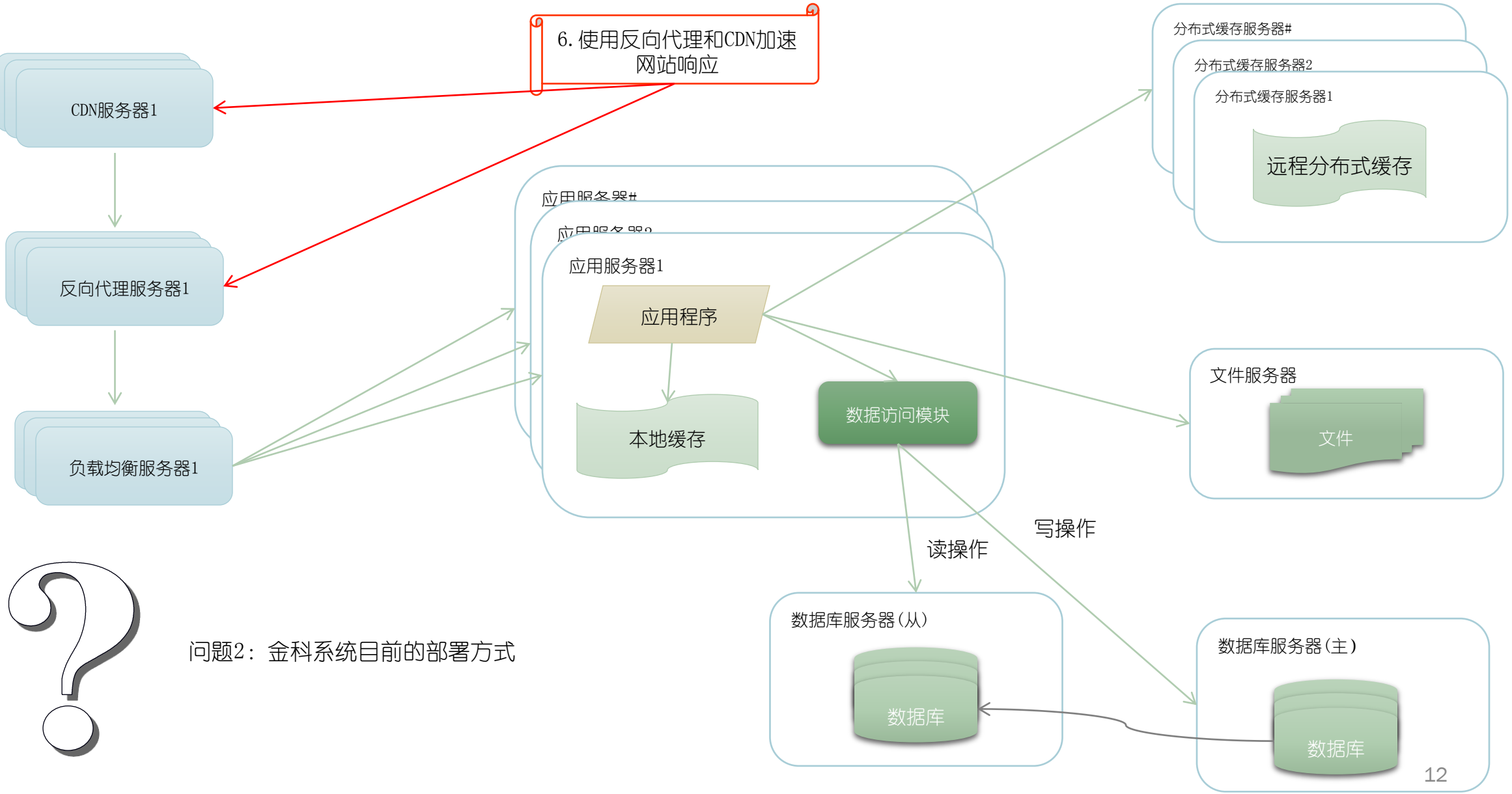
Jetty,Tomcat,Jboss,Weblogic,WebSphere
WEB容器， 应用服务器

架构演化



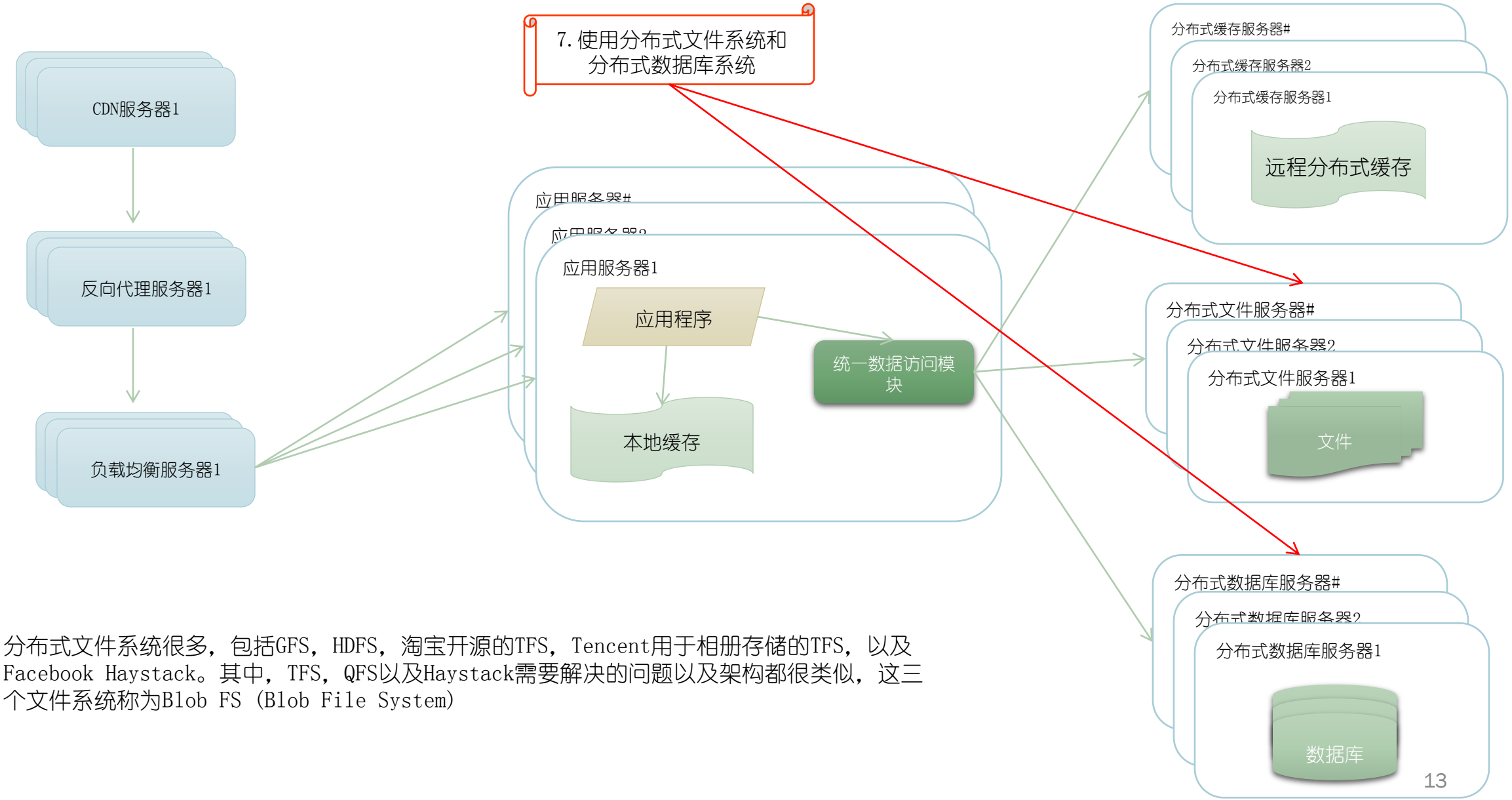
- ✓ Mysql proxy，监控SQL执行，数据流量，读写分离、数据切分、水平切分、负载均衡
- ✓ Amoeba(陈思儒,@冯大辉),基于XML的配置文件，用SQLJEP语法书写规则，比基于lua脚本的MySQL Proxy简单
- ✓ Cobar(Alibaba)，Cobar是在Amoeba基础上进化的版本，一个显著变化是把后端JDBC Driver改为原生的MySQL通信协议层。
- ✓ MyCat又是在Cobar基础上发展的版本(增加了对Order By、Group By、limit等聚合功能的支持)
- ✓ PDDL?

架构演化



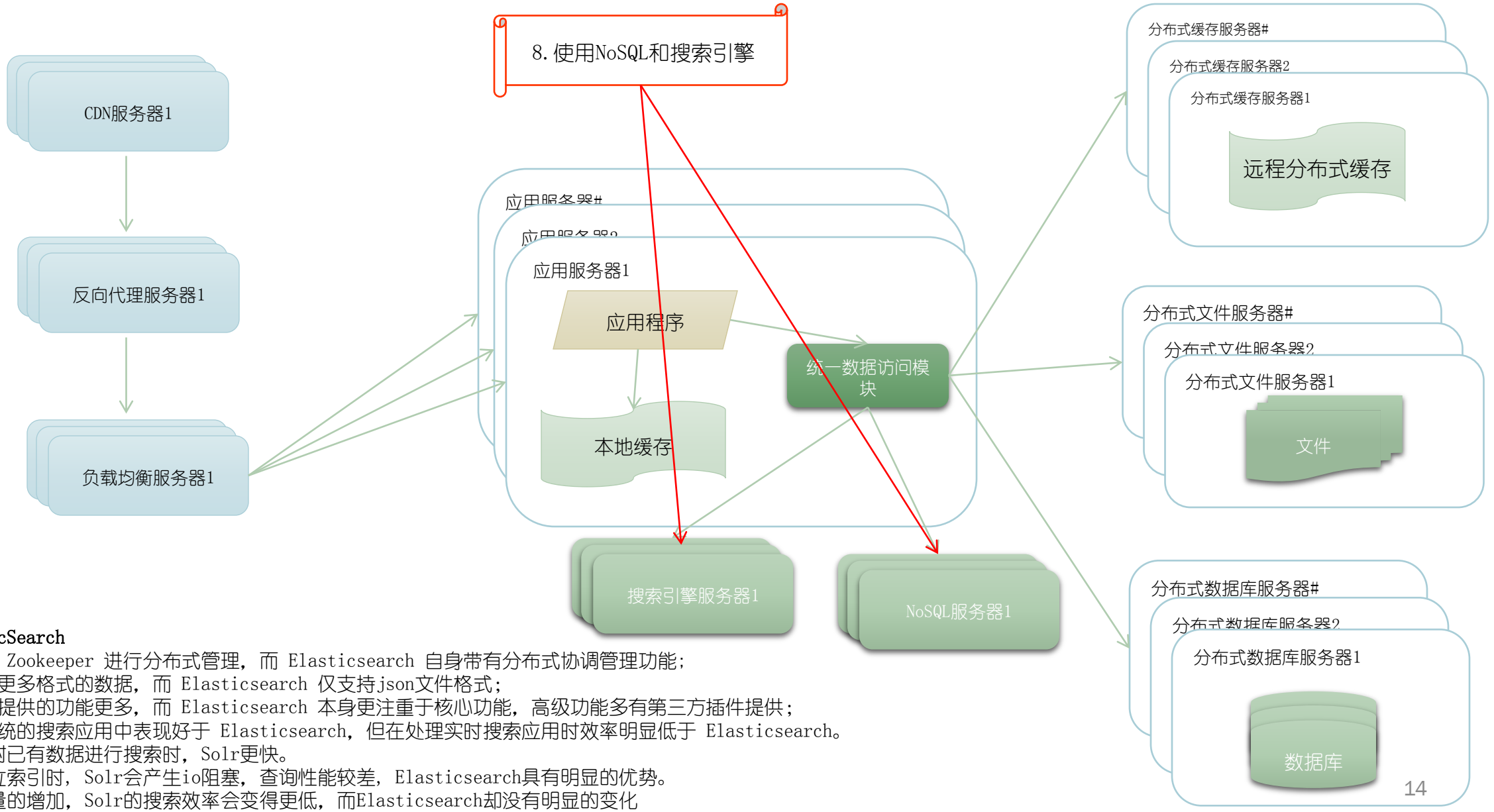
问题2：金科系统目前的部署方式

架构演化



分布式文件系统很多，包括GFS，HDFS，淘宝开源的TFS，Tencent用于相册存储的TFS，以及Facebook Haystack。其中，TFS，QFS以及Haystack需要解决的问题以及架构都很类似，这三个文件系统称为Blob FS（Blob File System）

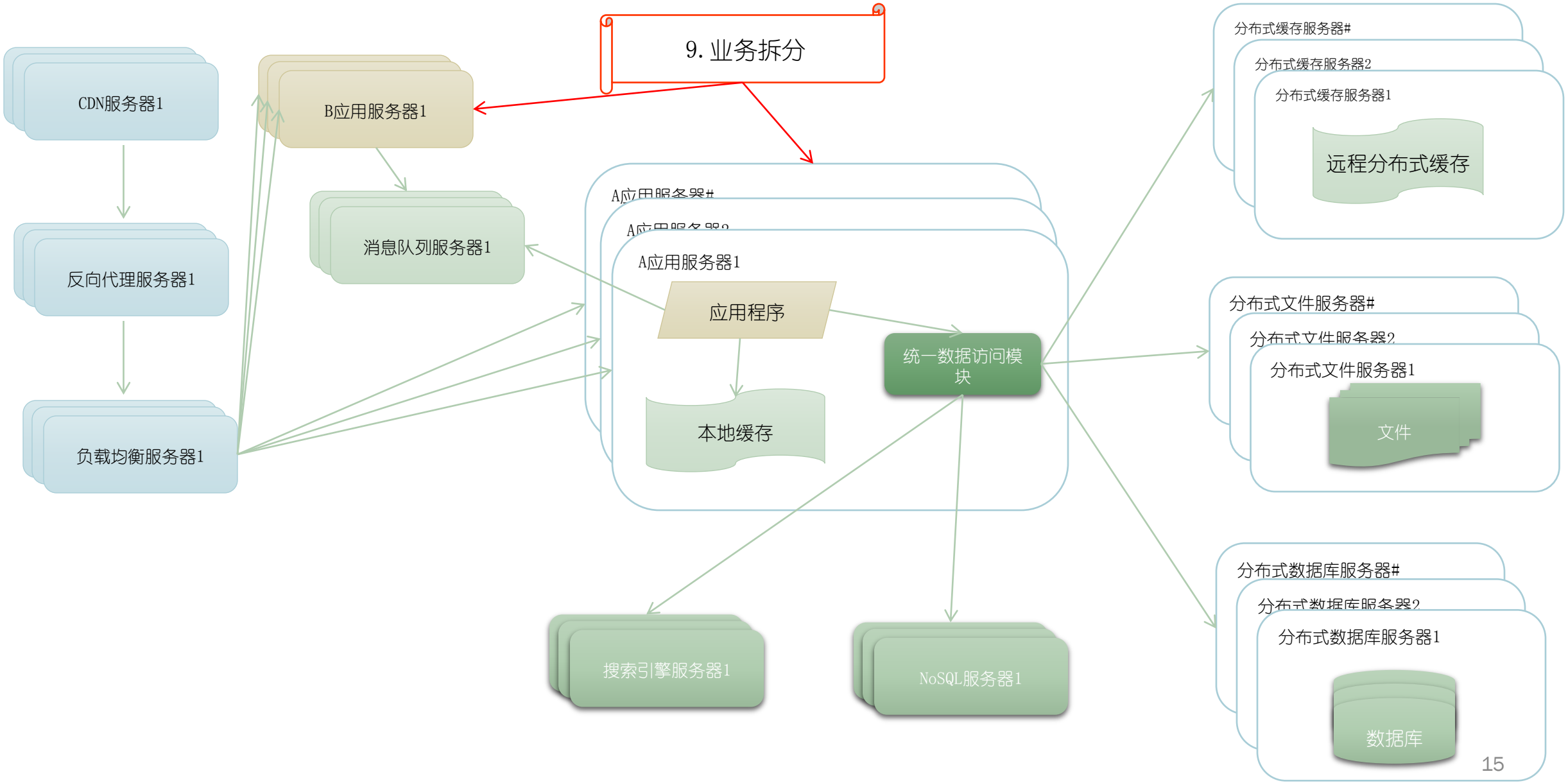
架构演化



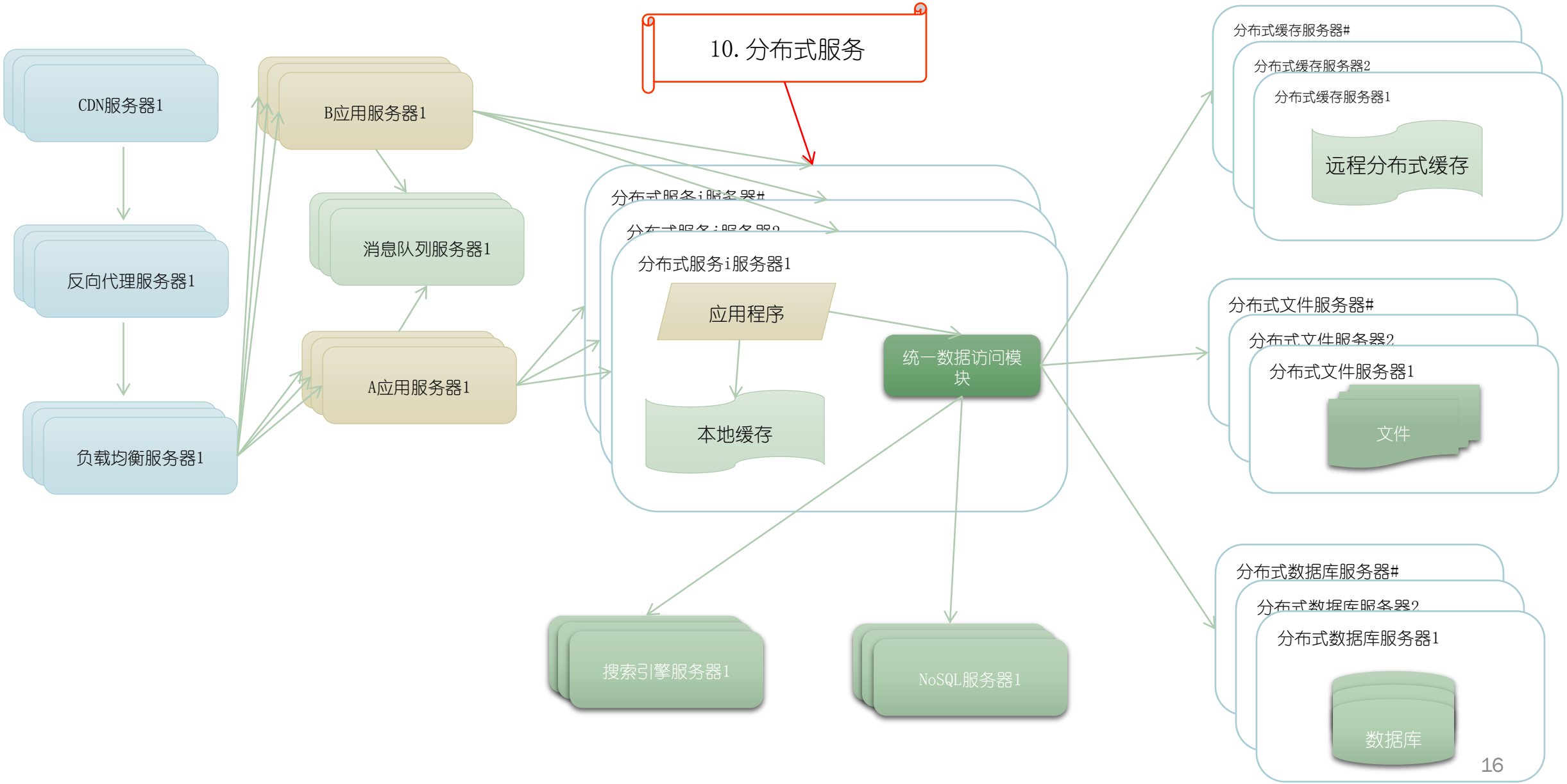
Solr, ElasticSearch

1. Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;
2. Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式;
3. Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供;
4. Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。
5. 当单纯的对已有数据进行搜索时，Solr更快。
6. 当实时建立索引时，Solr会产生io阻塞，查询性能较差，Elasticsearch具有明显的优势。
7. 随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化

架构演化


9. 业务拆分
CDN服务器1
反向代理服务器1
负载均衡服务器1
B应用服务器1
消息队列服务器1
A应用服务器1
应用程序
本地缓存
统一数据访问模块
搜索引擎服务器1
NoSQL服务器1
分布式缓存服务器#
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
分布式文件服务器#
分布式文件服务器1
文件
分布式数据库服务器#
分布式数据库服务器1
数据库

架构演化


10. 分布式服务
CDN服务器1
反向代理服务器1
负载均衡服务器1
B应用服务器1
消息队列服务器1
A应用服务器1
分布式服务i服务器#
分布式服务i服务器2
分布式服务i服务器1
应用程序
本地缓存
统一数据访问模块
搜索引擎服务器1
NoSQL服务器1
分布式缓存服务器#
分布式缓存服务器2
分布式缓存服务器1
远程分布式缓存
分布式文件服务器#
分布式文件服务器2
分布式文件服务器1
文件
分布式数据库服务器#
分布式数据库服务器2
分布式数据库服务器1
数据库

架构要素

性能：浏览器端，应用服务器端缓存，代码层面多线程，非阻塞IO等

可用性：HA，服务器宕机，请求可切换

伸缩性：集群，不断向集群加入服务器的手段来缓解不断上升的用户并发和不断增长的数据存储需求

扩展性：功能性需求，能否快速响应需求变化，主要手段-事件驱动架构和分布式服务

安全性：不受恶意访问和攻击，重要数据不被窃取

问题3：说出3种WEB前端性能优化方式？

- ✓ 减少HTTP访问次数
- ✓ 使用浏览器缓存，Cache-Control，Expires
- ✓ 启用压缩
- ✓ CSS放页面最上面，JS放在页面最下面
- ✓ 减少Cookie传输

架构模式

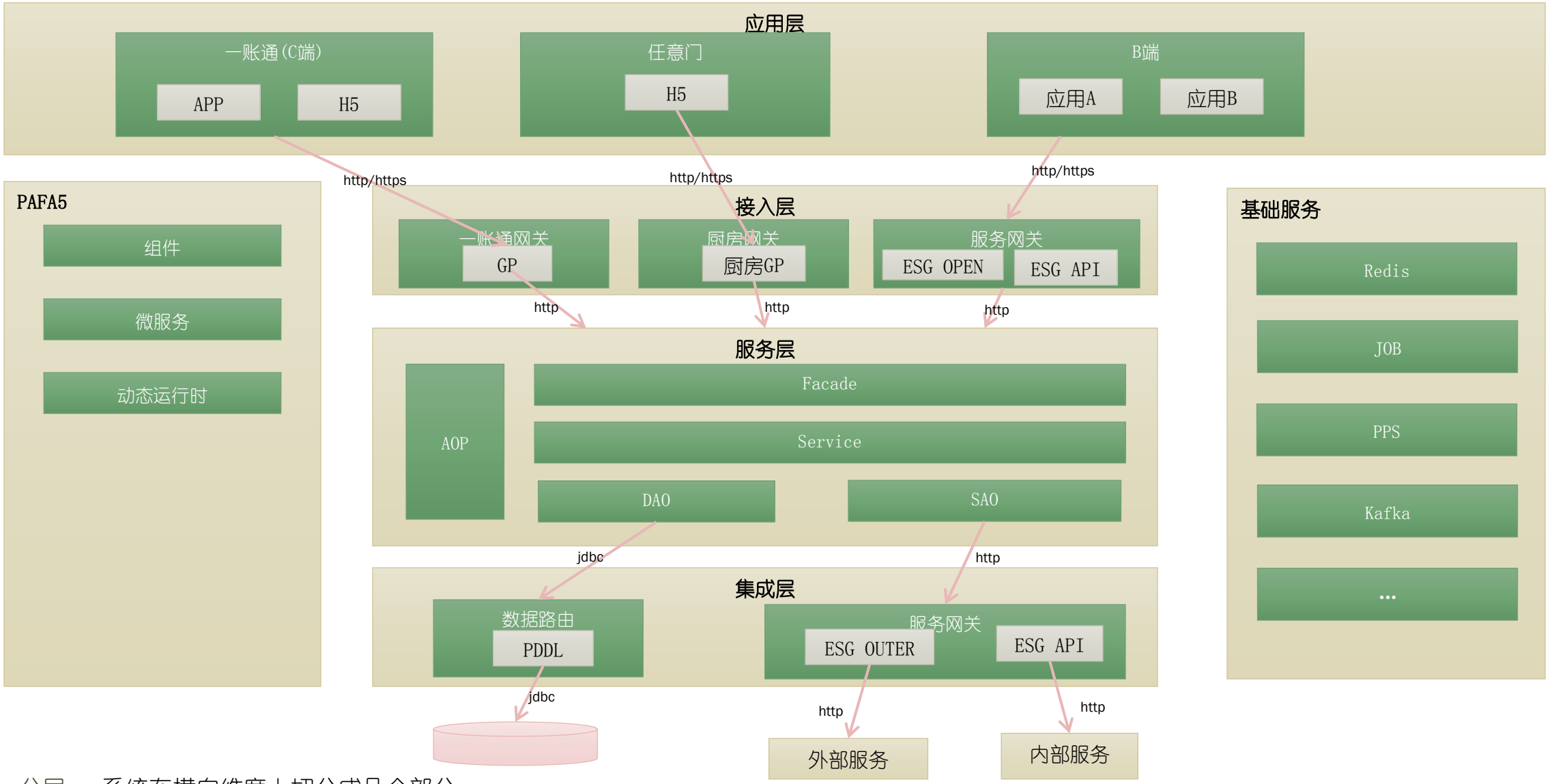
架构模式	设计模式
简单的说架构就是一个蓝图，是一种设计方案，将客户的不同需求抽象成为抽象组件，并且能够描述这些抽象组件之间的通信和调用	是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结，它强调的是一个设计问题的解决方法。
而架构是高层次的针对体系结构的一种设计思路，范畴比较大	设计模式主要是针对单一问题的解决方法，范畴比较小
分层,分割,分布式,集群,缓存,异步,冗余,自动化,安全	创建型 Factory Method（工厂方法）,Abstract Factory（抽象工厂）,Builder（建造者）, Prototype（原型）,Singleton（单例） 结构型 Adapter（适配器）,Bridge（桥接）,Composite（组合）,Decorator（装饰）, Facade（外观）,Flyweight（享元）,Proxy（代理） 行为型 Interpreter（解释器）,Template Method（模板方法）,Chain of Responsibility（责任链）,Command（命令）,Iterator（迭代器）,Mediator（中介者）,Memento（备忘录）,Observer（观察者）,State（状态）,Strategy（策略）,Visitor（访问者）

架构设计误区：
1. 一味追随大 公司的解决方案
2. 为了技术而技术，技术是为业务而存在的，除此毫无意义
3. 企图用技术解决所有问题，技术是用来解决业务问题的，而业务的问题，也可以通过业务的手段去解决
-12306

架构模式

分层	系统在横向维度上切分成几个部分
分割	系统在纵向方面对软件进行切分
分布式	不同模块部署在不同的服务器上，通过远程调用协同工作
集群	独立部署的服务器集群化
缓存	将数据放在距离计算最近的位置以加快处理速度
异步	生成者消费者不存在直接调用，通过共享数据的方式异步执行进行协作
冗余	服务器冗余运行，数据冗余备份
自动化	发布过程自动化，自动化代码管理，自动化测试，自动化安全检测，自动化部署，自动化监控，自动化失效转移，自动化降级
安全	密码，手机校验码，加密，加签，验证码

架构模式 - 分层



分层 - 系统在横向维度上切分成几个部分

架构模式 – 分割

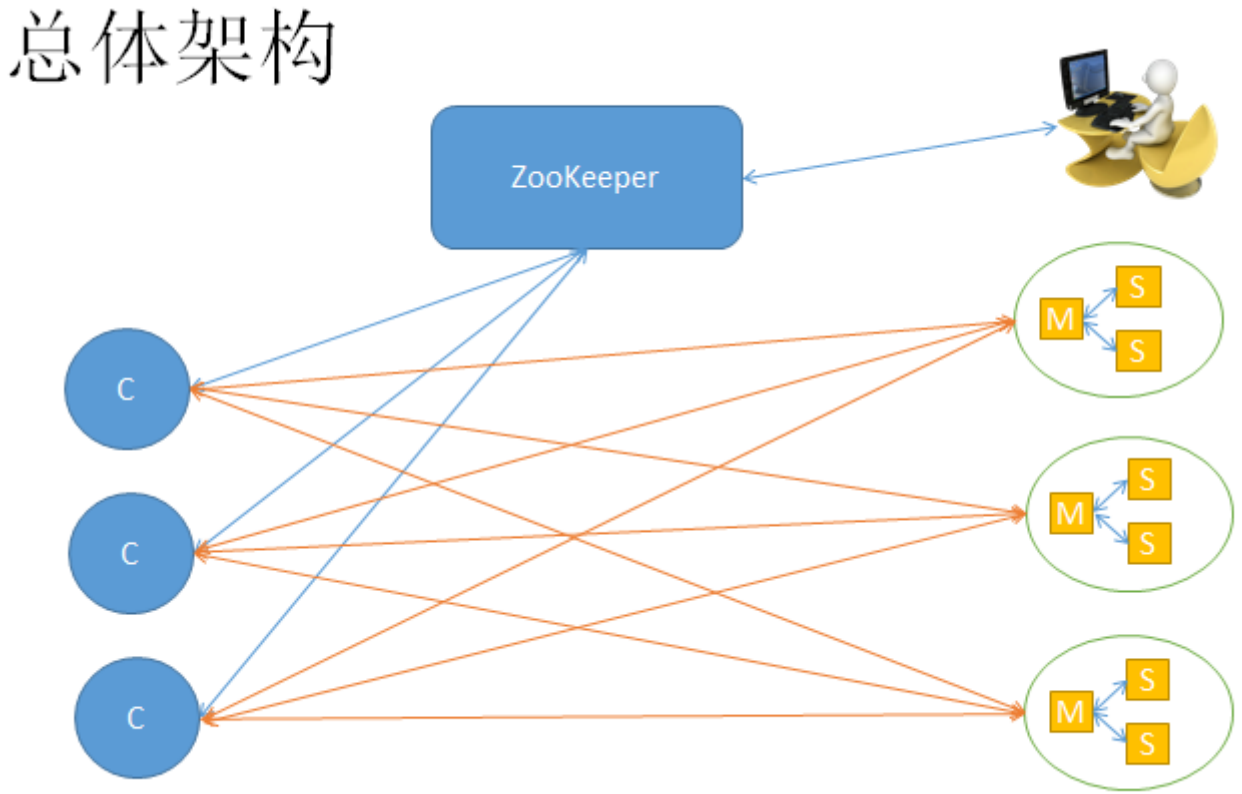
系统在纵向方面对软件进行切分

架构模式 - 分布式

不同模块部署在不同的服务器上，通过远程调用协同工作

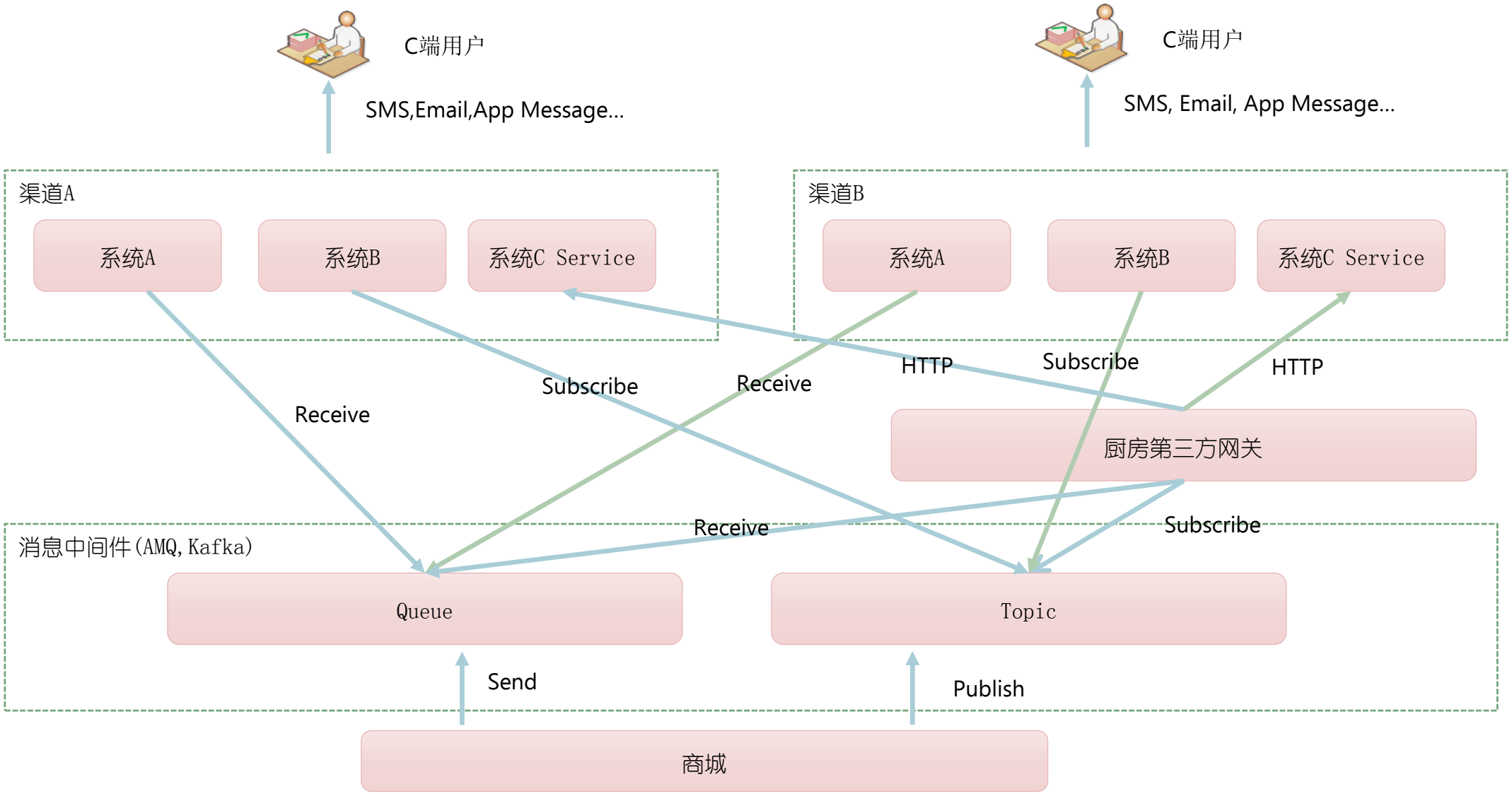
架构模式 - 集群

独立部署的服务器集群化



将数据放在距离计算最近/快的位置以加快处理速度

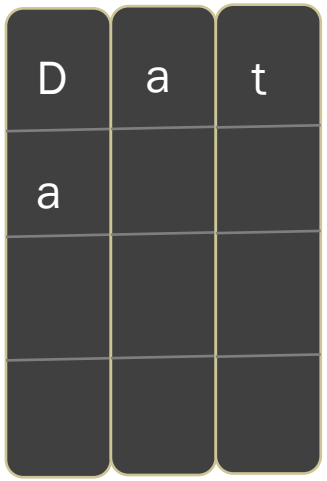
架构模式 - 异步



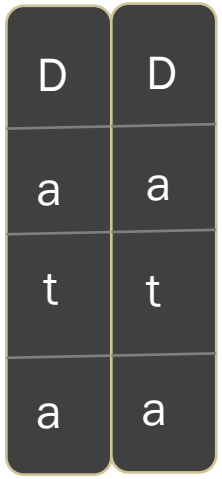
生成者消费者不存在直接调用，通过共享数据的方式异步执行进行协作

架构模式 - 冗余

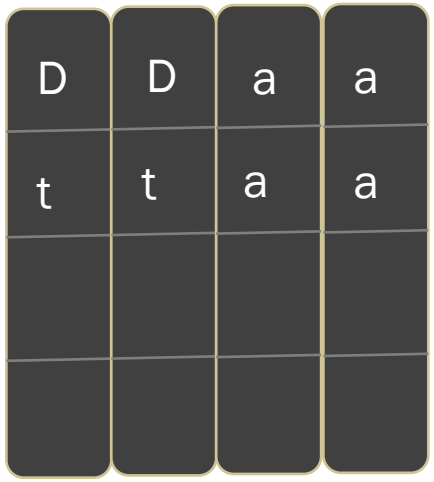
问题4：下面哪种是RAID5？



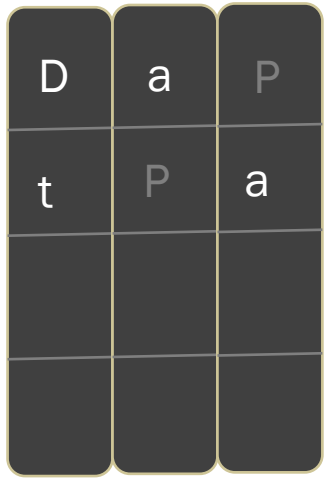
RAID 0



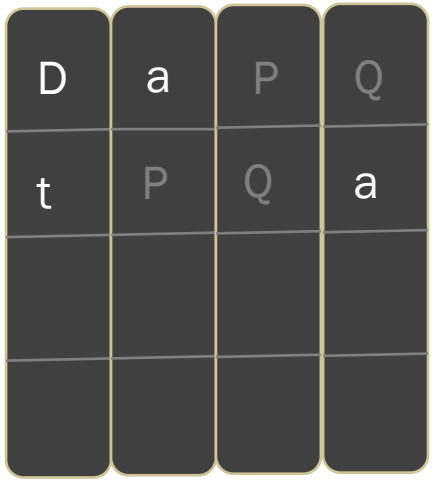
RAID 1



RAID 10

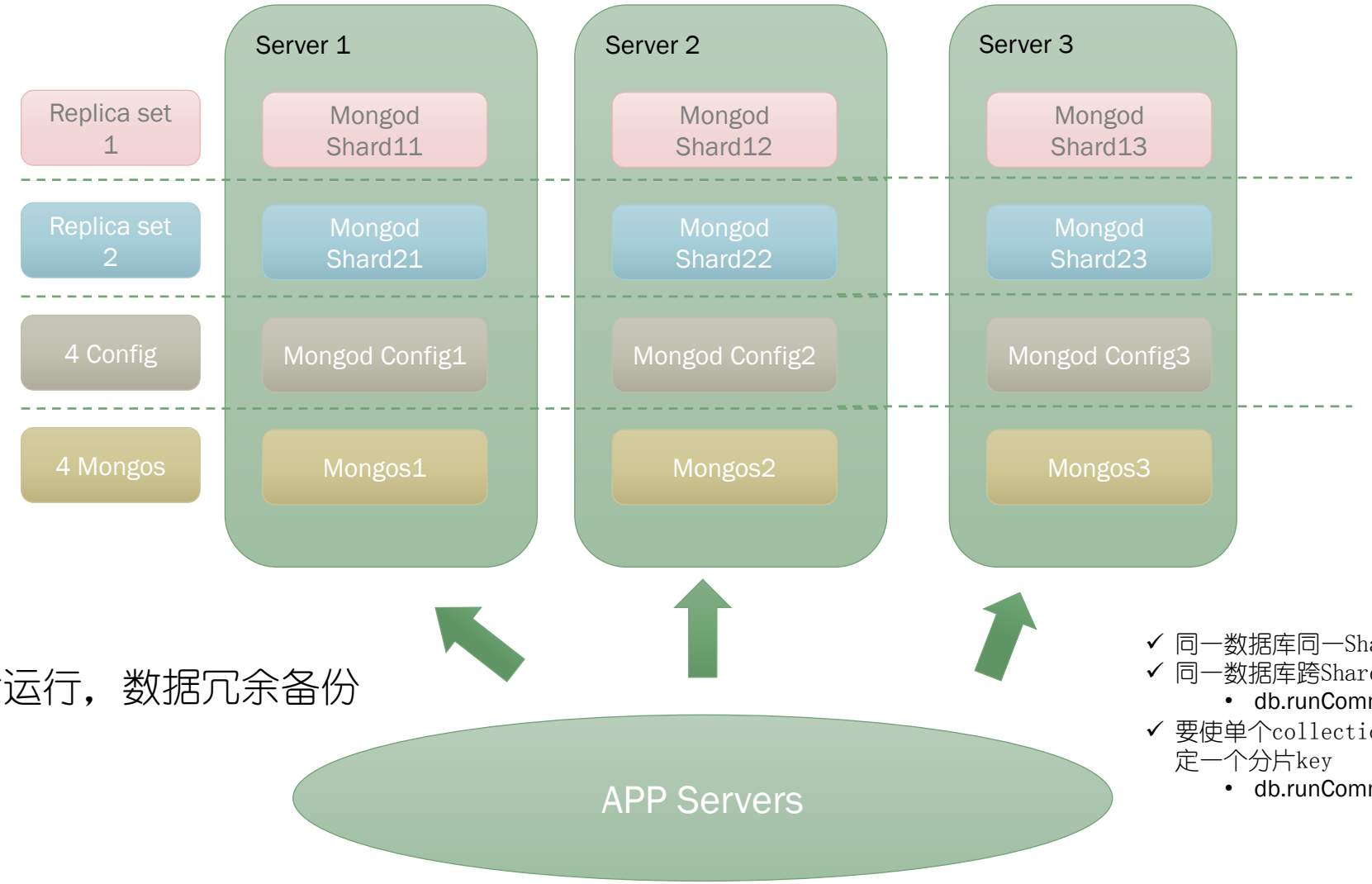


RAID 5



RAID 6

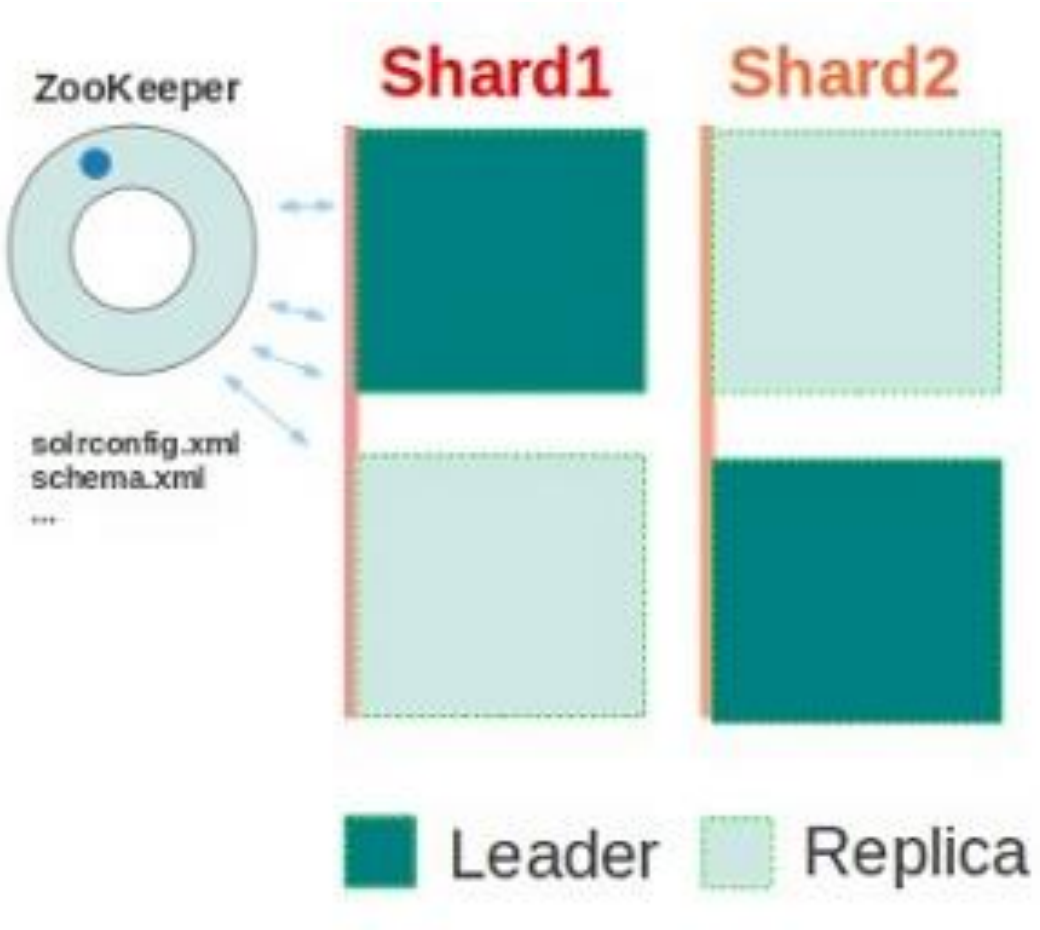
架构模式 – 冗余



服务器冗余运行，数据冗余备份

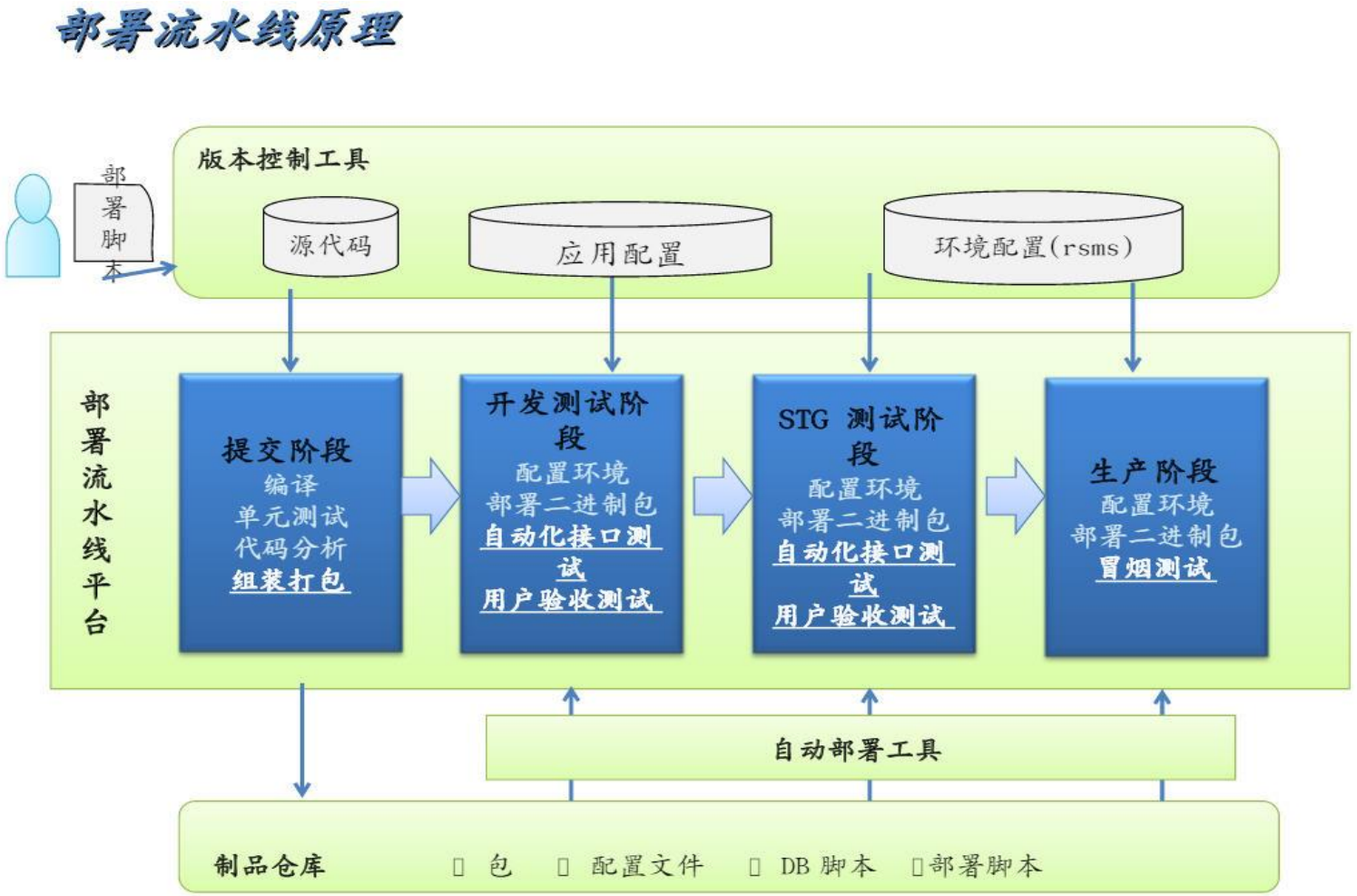
- ✓ 同一数据库同一Sharding
- ✓ 同一数据库跨Sharding
  - db.runCommand({ enablesharding : “” });
- ✓ 要使单个collection也分片存储，需要给collection指定一个分片key
  - db.runCommand({ shardcollection : “”,key : });

架构模式 - 冗余



服务器冗余运行，数据冗余备份 - Solr Cloud

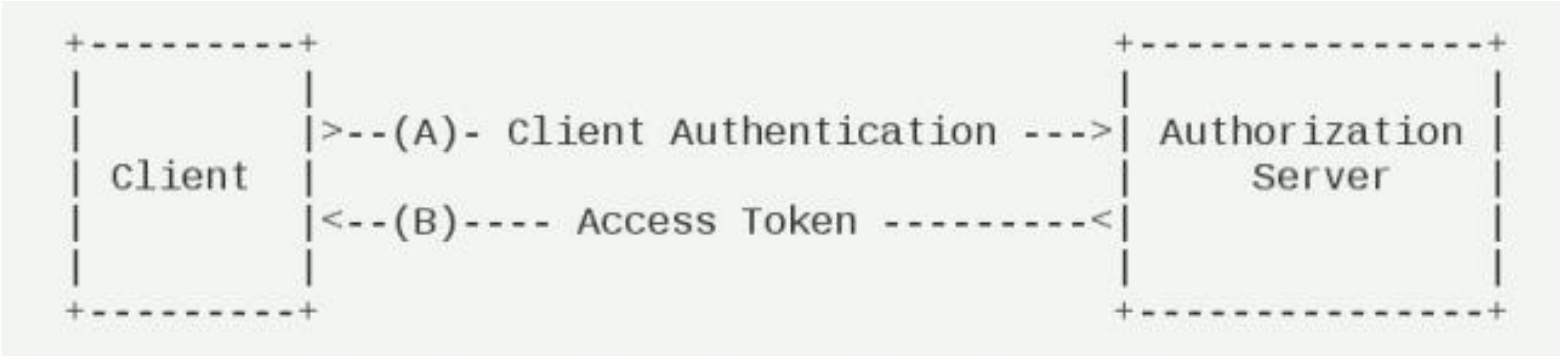
架构模式 – 自动化



发布过程自动化，自动化代码管理，自动化测试，自动化安全检测，自动化部署，自动化监控，自动化失效转移，自动化降级

问题5：加密和加签的区别

架构模式 - 安全



sign	String		【URI传参】API输入参数签名结果。外部调用，必填项。参考**签名算法**
sign_method	String		【URI传参】签名的摘要算法，默认为md5

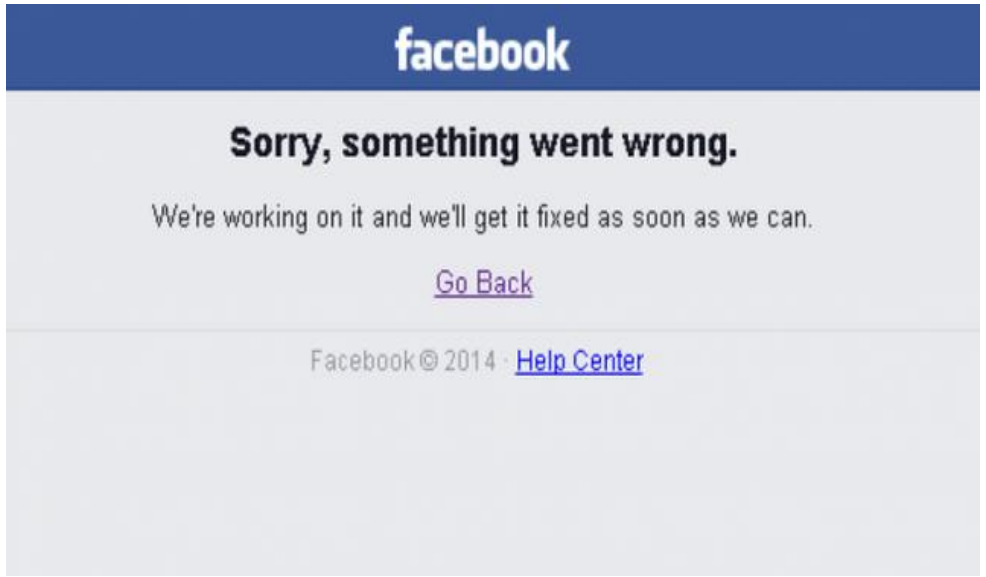
密码，手机校验码，加密，加签，验证码

高可用性架构定义

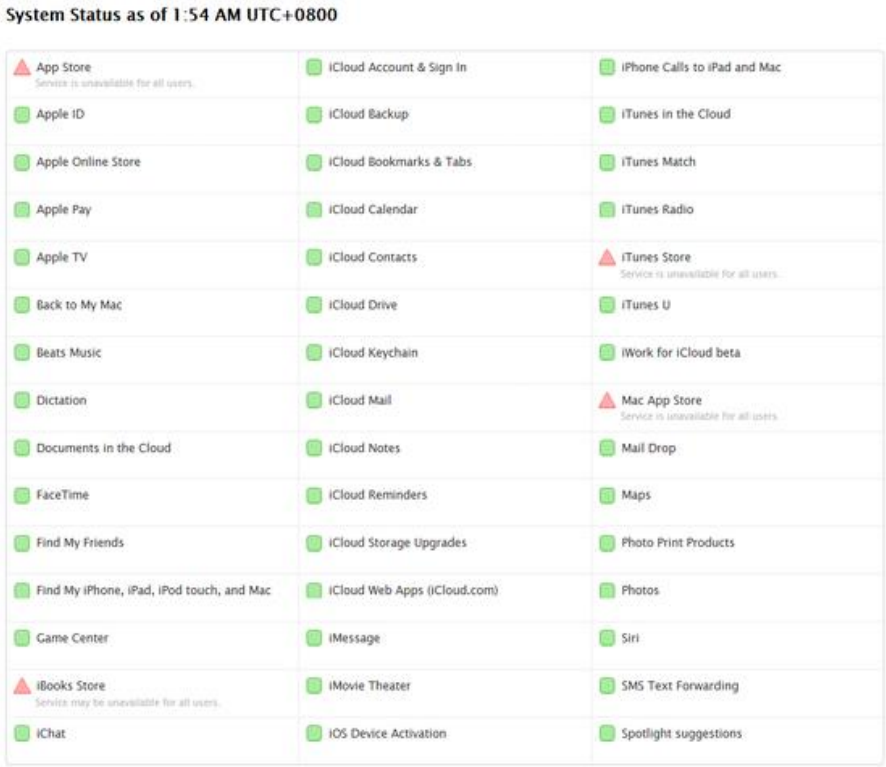
基础	不可用	高可用

• 硬件
  • 机器
  • CPU
  • 内存
  • 硬盘
  • 网卡
• 软件
  • 操作系统
  • 文件系统
  • 软件本身
• 网络
	• 硬件
	  • 生命周期
	  • 硬件故障
	• 软件
	  • 软件BUGS
	  • 软件间资源的争抢
	• 网络
		• 任何时间都正常提供读写服务
		  • 7*24H提供服务
		  • 机器硬件故障
		  • CPU、内存、硬盘、网卡
		  • 机器软件故障
		  • OS、FileSystem……

高可用性架构重要性

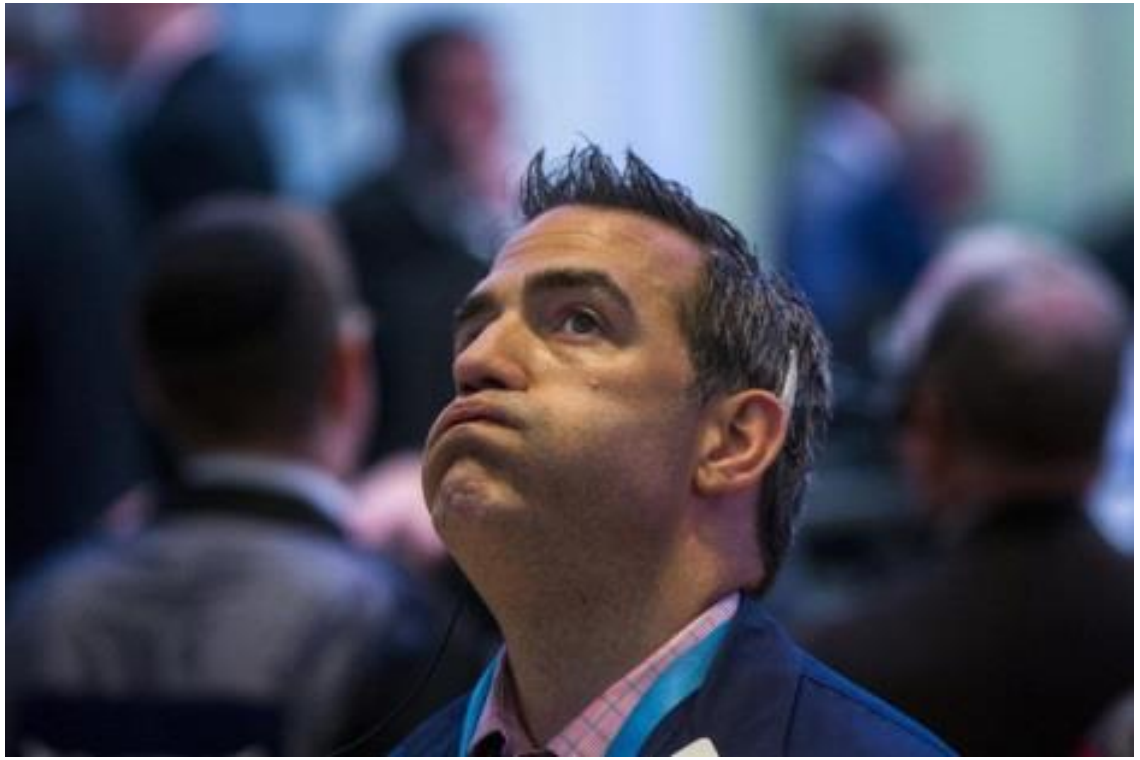


2015年1月27日，Facebook发言人称："此次故障与第三方攻击无关，发生故障的原因是我们对系统设置做了一点改动。"但相关数据显示，Facebook美国总部当天曾遭受大规模DDos攻击。



2015年3月11日，包括App Store、iTunes Store、Mac App Store以及iBooks Store在内的一系列苹果在线商店服务，遭遇大面积服务中断。据统计事故恢复时间长达11个小时。苹果公司针对该事件公开向用户道歉，并表示，宕机原因是苹果公司内部DNS错误。但此次故障，使苹果在股市上下跌1.82%，市值蒸发了130亿美元。

高可用性架构重要性



纽交所，2015年7月8日11时32分至15时10分，纽交所因故障暂停交易。据称，作为世界最大交易所之一，纽交所暂停交易超过3小时，可谓前所未有。随后，纽交所在推特上表示：此次故障由于交易所内部技术问题导致，已排除外部网络攻击的可能。纽交所主席法利表示，此次故障可能与软件升级有关。

高可用性架构重要性

2015年5月11日晚21时左右，网易旗下游戏、有道云笔记、LOFTER、考拉海购、网易公开课等无法正常访问。顿时谣言四起，有人称故障原因为：网易大厦着火！12日6时，大部分产品恢复正常。但此次网易骨干网络出现异常，也伤透了网易用户的心。游戏玩家称：一夜无事可做；正值创业期的闪电邮用户称：个人损失惨重；甚至还有用户直接宣布弃用网易产品。

2015年5月27日，支付宝大面积瘫痪，电脑端和移动端均无法进行转账付款，缘由是杭州市萧山区某地光缆被挖断，进而导致支付宝一个主要机房受影响，导致部分地区的支付宝服务中断数小时。

高可用性架构重要性

- 业务快速发展的支撑
- 系统不可用、系统频繁故障、系统不稳定
  - ✓ 用户体验差，用户留不住
  - ✓ 精力放在系统稳定性方面
  - ✓ 无精力响应业务功能需求
- 系统不可用，公司品牌、形象受影响
- 系统不可用，公司利益
  - ✓ 宕机时间==损失收入！（金钱！！）

高可用性架构手段

设计无状态化

- 应用服务器不保存业务的上下文信息
- 多个服务器实例之间完全对等
- 请求提交到任意服务器，处理结果完全一样

子系统冗余

- 服务器冗余运行
- 数据库冗余备份

幂等性设计

- 幂等性是指重复使用同样的参数调用同一方法时总能获得同样的结果。
- 天然有幂等性服务，如设置为男性。
- 转账需要通过交易号等信息进行服务调用有效性校验

异步调用

- 避免一个服务失败导致的整个服务失败
- 降低接口之间的耦合性
- 提高系统可用性
- 加快网站响应速度
- 消除并发访问高峰

超时机制

- 在应用程序中设置服务调用的超时时间，一旦超时，通信框架就抛出异常.

分级管理

- 运维上将服务器进行分级管理，核心应用和服务优先使用更好的硬件
- 部署上也进行必要的，避免故障的连锁反应
- 虚拟机隔离，物理机隔离，区域数据中心隔离

服务降级

- 高并发情况性能下降
- 两种手段，拒绝服务和关闭服务

服务治理

- 监控
  - 进程
  - 语义
  - 错误
  - 数据波动
- 服务可管理、可视化
- 日志监控系统

高可用性架构涉及环节

系统整体架构层面

- 硬件
- DNS
- CDN
- 接入层
- 逻辑层
- 数据存储层
- 分布式缓存层
- 数据层

架构关键节点层面

- 负载均衡
- 软件质量保证
- 预发布
- 灰度发布
- 安全
- 监控
- 回滚方案
- 线上问题定位分析

高可用性架构评价维度

-指系统在面对各种异常时可以提供正常服务的能力
-系统的可用性可以用系统停服务的时间和正常服务时间的比例来衡量
-系统不可用时间(故障时间=故障修复时间点-故障发现时间点)

可用性指标	可用性	说明
2个9的可用性（99%）	基本可用	一年停机的时间不能超过88个小时 365*24*60/100 = 88个小时
3个99的可用性（99.9%）	较高可用性	一年停机的时间不能超过9个小时 365*24*60/1000=9个小时
4个9的可用性（99.99%）	具备自动恢复能力的高可用	一年停机的时间不能超过53分钟 365*24*60/10000 = 53分钟
5个99的可用性（99.999%）	极高可用性	一年停机的时间不能超过5分钟 365*24*60/100000=5分钟

- 网站可用性
  - ✓ 较多的大网站可用性不足2个9，88小时
  - ✓ 国内的网站，厨房本身呢？
- 目标：做到4个9，具备自动恢复能力的高可用

高可用性架构分级

服务分级
•服务事故出现时，如何评估事故的影响面
•基于级别考核，可量化

服务级别	定义标准	服务名称
一级	服务每天PV达到5000w以上 收入达到公司在线收入的1/10 核心系统 后端基础服务至少为1个一级服务提供主要服务	列表系统，详情页系统 支付系统，订单系统,商品子系统 IDC网络，DNS
二级	服务每天PV达到1000w以上 收入达到公司在线收入的1/20 重要商务系统 后端基础服务至少为1个二级服务提供主要服务 公司内部核心信息系统	消息系统，个人中心，呼转服务 CRM，OA，BI系统 办公网络，VPN
三级	其他	其他

高可用性架构考核

事故级别定义
•确定服务级别
•使用最合适衡量标准确定事故的影响面

影响程序定级	影响程序低	影响程度中	影响程序高
对外完全停止服务时间	2-7分钟	7-30分钟	30分钟以上
系统对用户产生拒绝，占当日预期流量的比例	0.1%-1%	1%-6%	6%以上
系统返回结果不符合预期占当日总流量的比例	0.5%-2%	3%-15%	15%以上
受影响的用户占系统用户比例	3%-20%	20%-40%	40%以上
影响数据程序	部分数据丢失但能恢复	部分数据丢失不能恢复	所有数据丢失但不能恢复
收入损失：占日平均营收(线上加线下)比例	2%-5%	5%-30%	>30%
用户丢包率	5%-10%	10%-30%	>30%

事故定级	影响程序低	影响程度中	影响程序高
一级服务	严重	重大	特大
二级服务	一般	严重	重大
三级服务	N/A	一般	严重

高可用性架构为什么要分层

架构类型	特点	缺点与优点
ALL IN ONE 架构	整个架构只有一个模块，含数据部分、逻辑部分、接入部分、展示部分等	架构存在问题 耦合严重 职责不分明 模块庞大、臃肿 开发成本高、效率低下 运维成本高 组件间相互影响，一旦一个组件有问题，整个服务都受影响 扩展性差 性能极限差 牵一发而动全身
可高用分层架构	服务高可用需分层设计	-模块耦合性低 -模块职责分明 数据层、逻辑层、接入层、展示层等等 -模块间不再相互影响 -模块独立扩展 -系统整体性能高

高可用性架构分层原则

原则	说明
数据服务和逻辑服务分离	数据存储，业务逻辑
逻辑服务和接入服务分离	业务逻辑，接入层
接入服务和展示服务分离	接入层，数据展示
分层服务功能单一	数据，逻辑，接入，展示
分层间低耦合	接口交互
分层内高内聚	功能聚焦单一
分层适中	层次过多 -请求交互路径长 -请求响应延迟高 -层次多，运维成本高 -定位问题涉及层次多，定位复杂多增加，定位时间长 层次过少 -每个层次功能不单一，耦合性高 -模块内组件间相互影响高 -高可用无法保证

高可用性架构最佳实践

架构的分层取决于业务场景，MVC，三层，四层，五层,脱离业务场景谈架构分层绝对是耍流氓

业务发展阶段	说明
创业初期	✓ 满足业务快速发展 ✓ 可用性低 ✓ 分层少 ✓ ALL IN ONE
业务数据量、请求量快速增长期	✓ 引入分层 ✓ 接入层、逻辑层、数据存储等 ✓ 满足业务增长需求
业务请求高并发，海量存储期	✓ 每层进一步细化 ✓ 分布式存储、NoSQL、RDBMS分库分表
业务多、请求多、关系复杂	✓ 服务化 ✓ 解耦、稳定

谢谢