

# 架构师笔记-下

---

## 34. 深入理解微服务架构：银弹or焦油坑

### 微服务与SOA的关系

#### 1、微服务和SOA的一些具体用法

##### (1) 服务粒度

整体上来说，SOA的服务粒度要更粗一些，而微服务的服务力度要更细一些。

##### (2) 服务通信

SOA采用ESB作为服务间通信的关键组件，负责服务定义、服务路由、消息转换、消息传递，总体上是重量级的实现。微服务推荐使用统一的协议格式。（smart endpoints and dumb pipes。）

##### (3) 服务交付

SOA的服务交付并没有特殊要求，因为SOA更多的是考虑兼容已有系统；微服务架构的理念要求“快速交付”，相应地要求采用自动化测试、持续集成、自动化部署等敏捷开发相关的最佳实践。如果没有这些基础能力支撑，微服务规模一旦变大，整体就难以达到快速交付的要求，这也是很多企业在实行微服务时踩过的一个明显的坑，就是系统拆分为微服务后，部署的成本呈指数级增长。

##### (4) 应用场景

SOA更适合庞大、复杂、异构的企业级系统，这也是SOA诞生的背景。微服务更加适合快速、轻量级、基于web的互联网系统，这里系统业务变化快，需要快速尝试，快速交付。

对比维度	SOA	微服务
服务粒度	粗	细
服务通信	重量级, ESB	轻量级, 例如, HTTP RESTful
服务交付	慢	快
应用场景	企业级	互联网

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.

( <https://martinfowler.com/articles/microservices.html> )

Small、lightweight、automate，基本上浓缩了微服务的精华，也是微服务与SOA的本质区别。

## 微服务的陷阱

- 1、服务划分过细，服务间关系复杂
- 2、服务数量太多，团队效率急剧下降。
- 3、调用链太长，性能下降。
- 4、调用链太长，问题定位困难。
- 5、没有自动化支撑，无法快速交付。
- 6、没有服务治理，微服务多了后管理混乱。

## 35. 微服务架构最佳实践：方法篇

### 1、服务粒度

“三个火枪手”原则应用于微服务设计和开发阶段，如果微服务经过一段时间后已经比较稳定，处于维护期了，无须太多的开发，那么平均每个人维护一个微服务甚至几个微服务都是可以的。当然，考虑到人员备份问题，每个微服务最好安排两个人进行维护，每个人都可以维护多个微服务。

### 2、拆分方法

#### (1) 基于业务逻辑进行拆分

这是最常见的一种拆分方式，将系统中的业务模块按照职责范围识别出来，每个单独的业务模块拆分为一个服务。但是按照业务拆分一般没有很好的标准，最终都会根据“三个火枪手原则”处理。

#### (2) 基于可扩展性进行拆分

将系统中的模块按照稳定性排序，将已经成熟和改动不大的服务拆分为稳定服务，将经常变化和迭代的服务拆分为变动服务。稳定服务的粒度可以粗一些，即使逻辑上没有强关联的系统一个可以放在一个服务中。不稳定服务的粒度可以细一些，但也不要太细。

#### (3) 基于可靠性拆分

将系统中的模块按照优先级排序，将可靠性要求高的核心服务和可靠性要求低的非核心服务拆分开来，然后重点保障核心服务的高可用。

这样拆分带来如下介个好处：

- 避免非核心服务故障影响核心服务
- 核心服务服务高可用方案可以更简单
- 能够降低高可用成本

#### (4) 基于性能拆分

基于性能拆分和基于可靠性拆分类似，将性能要求高的或者性能压力大的模块拆分出来，避免性能压力大的服务影响其他服务。

### 3、基础设施

大部分人关注的是微服务的”small“和”lightweight“特性，但实际上真正决定微服务成败的，恰恰是那个被大部分人都忽略的”automated“。因为如果基础设施不健全，那微服务就是焦油坑，让研发、测试、运维都陷入到陷阱中。



基础设施搭建的优先级：

- (1) 服务发现、服务路由、服务容错：这是最基本的微服务基础设施。
- (2) 接口框架、API网管：主要是为了提升研发效率，接口框架是提升内服的研发效率，API网关是为了提升与外部对接的效率。
- (3) 自动化部署、自动化测试、配置中心：主要是为了提升测试和运维效率。
- (4) 服务监控、服务跟踪、服务安全：主要是为了进一步提升运维效率。

## 36.微服务架构最佳实践：基础设施篇

### 1、自动化测试

微服务将原本大一统的系统拆分为多个独立运行的“微”服务，微服务之间的接口数量大大增加，并且微服务提倡快速交付，版本周期短，版本更新频繁。如果每次更新都靠人工来回归整个系统，则工作量大，效率低下，达不到“快速交付”的目的，因此必须通过自动化测试系统来完成绝大部分测试回归工作。

自动化测试的范围包括代码级的单元测试，单个系统的集成测试，系统间的接口测试，理想情况下是每类测试都自动化。如果无法做到全面覆盖，至少做到接口测试自动化。

### 2、自动化部署

自动化部署包括版本管理、资源管理（例如，机器管理、虚拟机管理）、部署操作，回退操作等。

### 3、配置中心

### 4、接口框架

统一接口协议，统一接口传递数据的格式。

### 5、API网关

API网关是外部系统访问的接口，所有的外部系统都需要通过API网关来接入系统。API网关主要包括接入鉴权、权限控制、传输加密、请器路由、流量控制等功能。

### 6、服务发现

服务发现主要有两种方式：自理 和 代理

### 7、服务路由

有了服务发现之后，微服务之间能够方便的获取相关的配置信息，但具体进行某次请求时，我们还需要从符合条件的可用微服务节点中选择一个具体的节点发起请求，这就是服务路由需要完成的功能。

### 8、服务容错

9、服务监控

10、服务跟踪

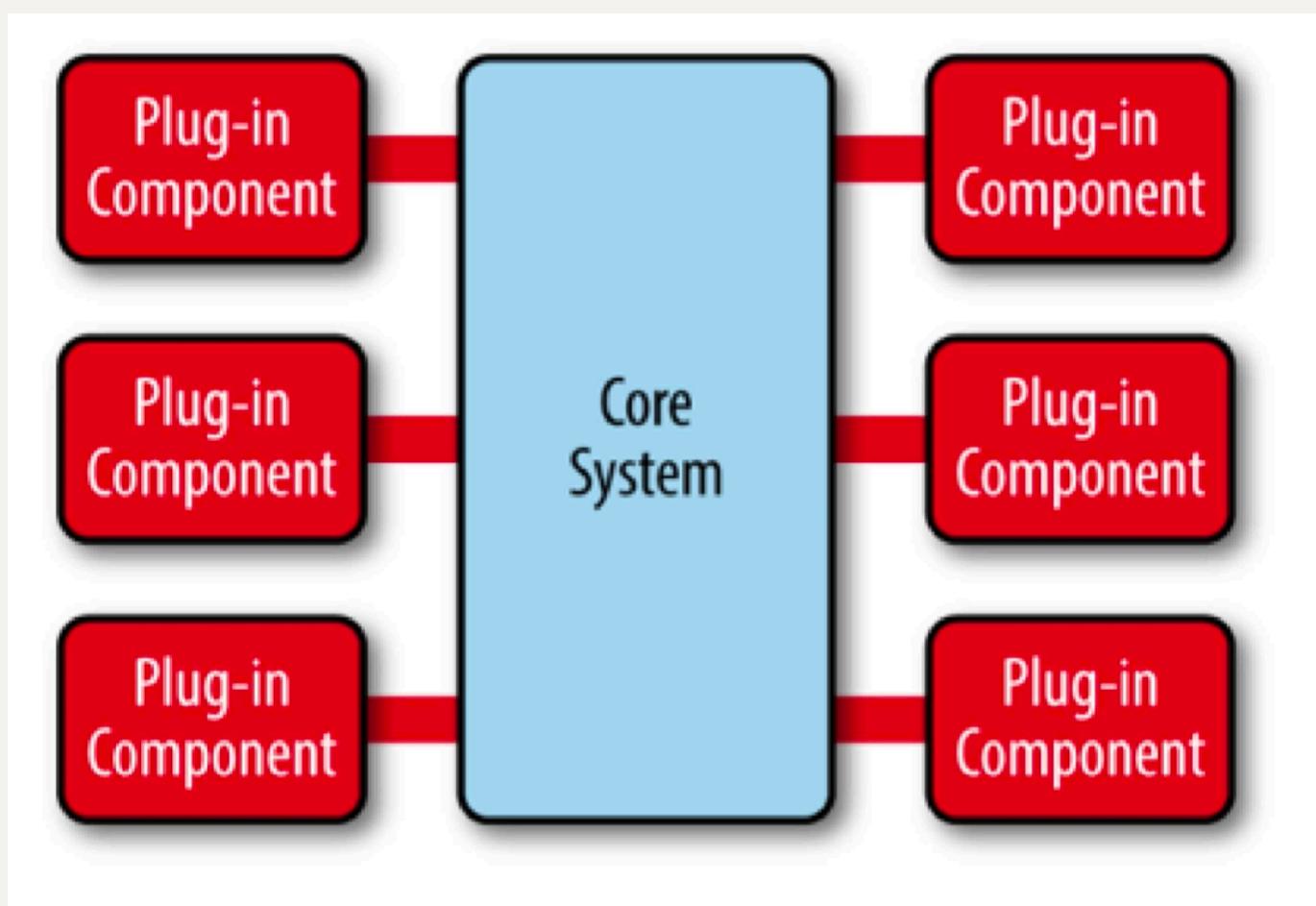
11、服务安全

## 37.微内核架构详解

微内核架构也被称为插件化架构，是一种面向功能进行拆分的可扩展架构，常用于实现基于产品的应用（只存在多个版本，需要通过下载才能使用的）。

### 1、基本架构

微内核架构包含两类组件：核心系统和插件模块。核心系统负责和具体业务无关的功能，例如模块加载、模块通信等；插件模块负责实现具体的业务逻辑。



### 2、设计关键点

微内核的核心设计技术有：插件管理、插件链接和插件通信。

### (1) 插件管理

核心系统需要知道有哪些插件可用，如何加载这些插件，什么时候加载插件。常见的实现方法是插件注册表机制。

### (2) 插件链接

通常来说，核心系统必须制定插件和核心系统的链接规范，然后插件按照规范实现，核心系统按照规范加载即可。

常见的链接机制有OSGI、消息模式、依赖注入等。

### (3) 插件通信

插件通信指插件间的通信，虽然设计插件的时候是完全解耦的，但实际业务运行过程中，必然会出现某个业务流程需要多个插件协作的场景，这就需要多个插件进行通信。由于插件之间没有联系，通信必须通过核心系统，因此核心系统必须提供通信机制。

## 3、OSGI框架简介

OSGI是一个插件化的标准，而不是一个可运行的框架，Eclipse采用的是Equinox，类似的还有Apache的Felix、Spring的Spring DM。

OSGI的逻辑架构如下：

**Service**

**Service registry**

**Lifecycle**

**BundleActivator**

**BundleContext**

**Module**

**Bundle**

**Bundle**

### (1) 模块层

模块层实现插件管理功能，OSGI中插件被称为Bundle，每个Bundle是一个java的jar文件，每个Bundle中都包含一个元数据文件MANIFEST.MF，这个文件包含了Bundle的基本信息。

### (2) 生命周期层

生命周期层提供插件链接功能，提供了执行时的模块管理、模块对底层OGSI框架的访问。生命周期层精确定义了Bundle生命周期的操作（安装、更新、启动、停止、卸载），Bundle必须按照规格实现各种操作。

### (3) 服务层

服务层实现插件通信的功能。提供了一个服务注册功能，通过服务中心注册自己，搜索要使用的服务。

## 4、规则引擎架构简介

规则引擎架构从结构上来看也是一种微内核架构，其中执行引擎可以看做是微内核。

规则引擎有如下特点：

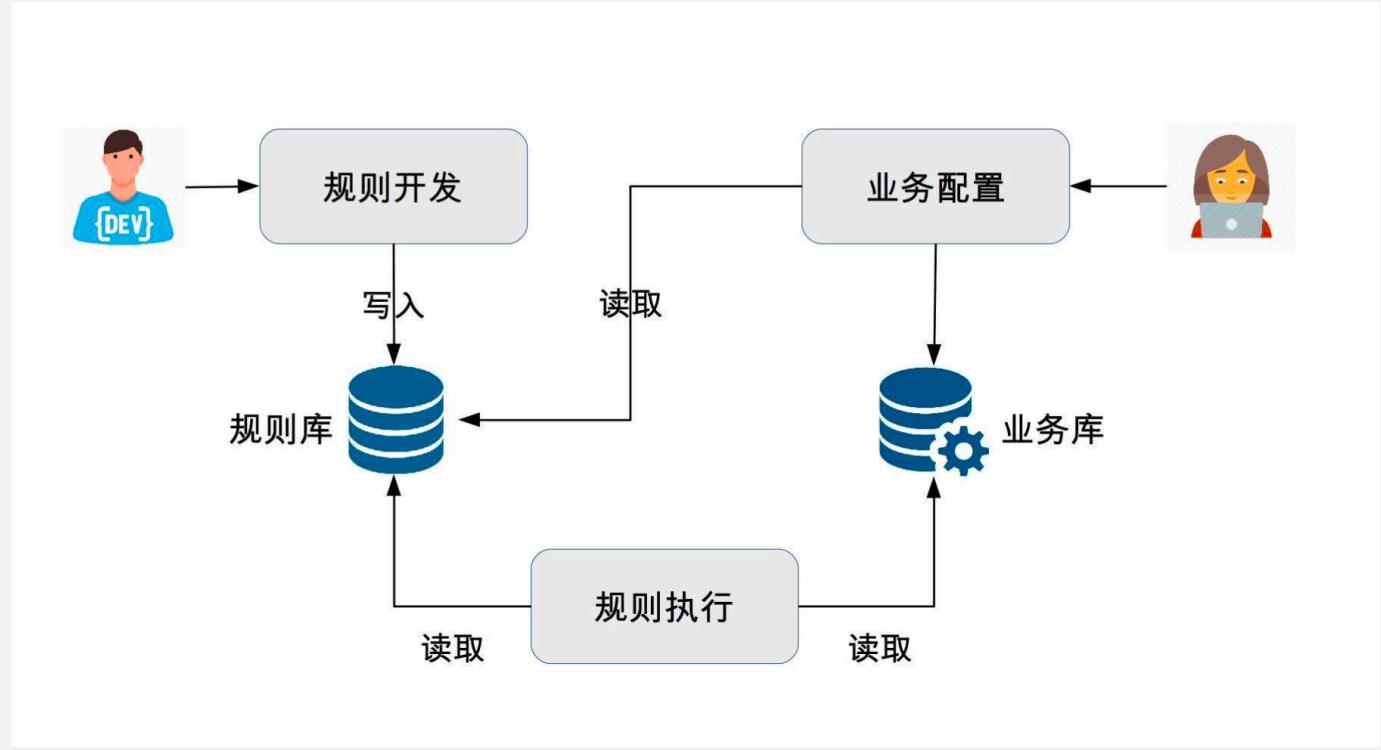
### (1) 可扩展

业务逻辑实践与业务系统分离，可以在不改变业务系统的情况下扩展新的业务功能。

### (2) 易理解

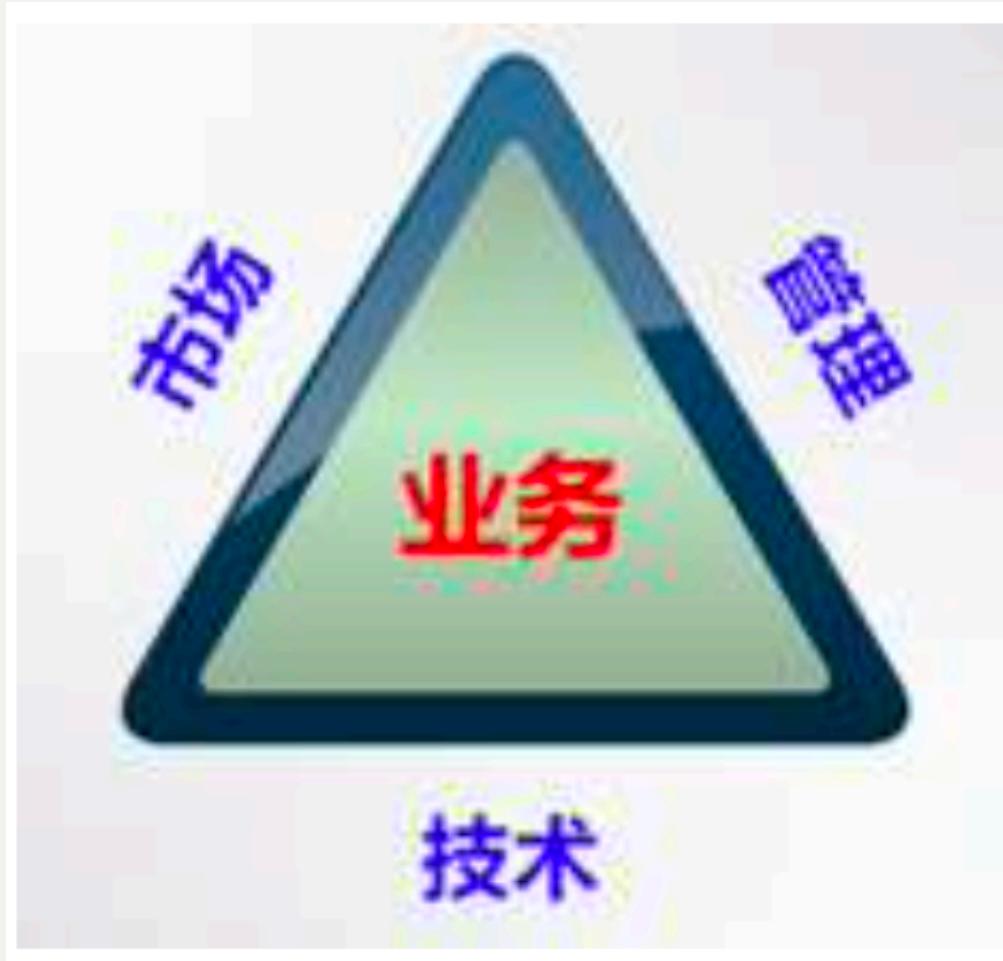
规则通过自然语言描述，业务人员易于理解和操作，而不像代码那样只有程序员才能理解和开发。

### (3) 高效率



- 开发人员将业务系统分解提炼成多个规则，将规则保存在规则库中。
- 业务人员根据业务需求对业务进行排列组合。
- 规则引擎执行业务流程。

## 38. 架构师应该如何判断技术演进方向



业务是核心，而我们可以将业务简单的分为两类：产品类和服务类。

对于产品类的业务，技术创新能够推动业务发展。而对于服务类产品，一般是业务发展来推动技术创新。因为对于服务类产品，用户一般是根据规模来进行选择。

总结：除非是开创新的技术能够推动或者创造一种新的业务，其他情况下，都是业务的发展推动了技术的发展。

## 39. 互联网技术的演技模式

互联网业务发展一般分为如下几个阶段：初创期、发展期、竞争期、成熟期。

不同时期的差别主要体现在两个方面：复杂性、用户规模。

# 业务复杂性

互联网业务发展第一个主要方向就是”业务越来越复杂“，我们来看看不同时期业务复杂性的表现：

## 1、初创期

初创期的业务对速度的要求就一个字”快“，但这个时候又是业务团队最弱小的时候，所以为了应对一般就是能买就买，能用开源就用开源。

## 2、发展期

对于绝大部分技术团队来说，这个阶段技术核心的工作就是快速实现各种需求，只有这样才能够满足业务发展的需求。

一般会经历如下几个阶段：

### (1) 堆功能期

业务进入快速发展的初期，此时团队规模也不大，业务需求又很紧张，最快实现业务需求的方式是继续在原有系统中不断地增加新的功能，重构、优化、架构方面的工作即使想做，也会受制于人力和业务发展的压力而放在一面。

### (2) 优化期

经过了堆功能期的迅猛发展，功能越来越多、系统变的越来越复杂。一个典型的场景就是做一个功能要改很多地方，一不小心就改出很多问题，这时就会使用重构等思想来对系统进行优化。

### (3) 架构期

架构期可用的手段很多，但归根结底可以总结为一个字”拆“，什么地方都可以拆。

拆功能：例如，将购物系统拆分为登录认证子系统、订单系统、查询系统和分析系统等。

拆数据库：MySQL一台变两台，两台变四台，增加DBProxy、分库分表等。

拆服务器：服务器一台变两台、两台变四台，增加负载均衡的系统，如Nginx、HAProxy等。

### 3、竞争期

由于竞争的压力，对技术的要求已经更上一层楼了。

随着新功能的增加和系统的不断拆分，技术工作又开始进入了“慢”的状态。这是为什么呢？因为量变带来了质变，主要体现在如下几个方面：

#### (1) 重复造轮子

系统越来越多，各个系统相似的工作越来越多。

#### (2) 系统交互一团乱麻

系统越来越多，各系统的交互关系变成了网状。

主要的解决手段有：

#### (1) 平台化

目的在于解决“重复造轮子”的问题。

#### (2) 服务化

目的在于解决系统交互的问题，

### 4、成熟期

这个阶段一般就是进行各种优化。

### 用户规模

用户量增大对技术的影响主要体现在两个方面：性能要求越来越高、可用性要求越来越高。

## 量变带来质变

阶段	用户规模	业务阶段	技术影响
婴儿期	0 ~ 1万	初创期	用户规模对性能和可用性都没有什么压力，技术人员可以安心睡好觉
幼儿期	1万 ~ 10万	初创期	用户规模对性能和可用性已经有一点压力了，主要体现为单台机器（服务器、数据库）可能已经撑不住了，需要开始考虑拆分机器，但这个时候拆分还比较简单，因为机器数量不会太多
少年期	10万 ~ 100万	发展期	用户规模对性能和可用性已经有较大压力了，除了拆分机器，已经开始需要将原来大一统的业务拆分为更多子业务了
青年期	100万 ~ 1000万	竞争期	用户规模对性能和可用性已经有很大压力了，集群、多机房等手段开始用上了。虽然如此，技术人员还是很高兴的，毕竟到了此时公司已经发展得非常不错了
壮年期	1000万 ~ 1亿	竞争期 & 成熟期	用户规模对性能和可用性已经有非常大压力了，可能原有的架构和方案已经难以继续扩展下去，需要推倒重来。不过如果你真的身处这样一个公司，虽然可能有点辛苦，但肯定会充满干劲，因为这样的机会非常难得，也非常锻炼人
巨人期	1亿 +	成熟期	和壮年期类似，不过如果你真的身处这样一个公司，虽然可能有点辛苦，但估计做梦都要笑醒了！因为还没有哪个行业能够同时容纳两家1亿+用户的公司

## 40. 互联网架构模板：“存储层”技术



## SQL

数据如何拆分，数据如何组合，是一个复杂的问题，如果每个业务系统都独立实现一次，重复造轮子将造成投入浪费，效率降低，业务开发想快起来都难。

所以互联网公司的流行做法是，在业务发展到一定阶段后，就将这部分功能独立成中间件。但是这样的工作对技术要求很高，一般都是大公司来做。中小公司一般会采用开源解决方案。

随着业务继续发展，那么每个系统使用中间件部署自己的集群，又会导致数据库资源利用率不高，各SQL集群分开维护成本高的问题，这是实力雄厚的公司就会搭建SQL存储平台，以对业务透明的方式提供资源分配、数据备份、迁移、容灾、读写分离、分库分表等一系列服务。

## NoSQL

由于NoSQL本身就带有集群方案，所以大多数公司不会一开始就将其包装为存储平台。而是在发展到一定规模后才会在NoSQL集群的基础上建立存储平台，统一存储平台主要实现以下几个功能：

1、资源按需动态分配

2、资源自动化管理

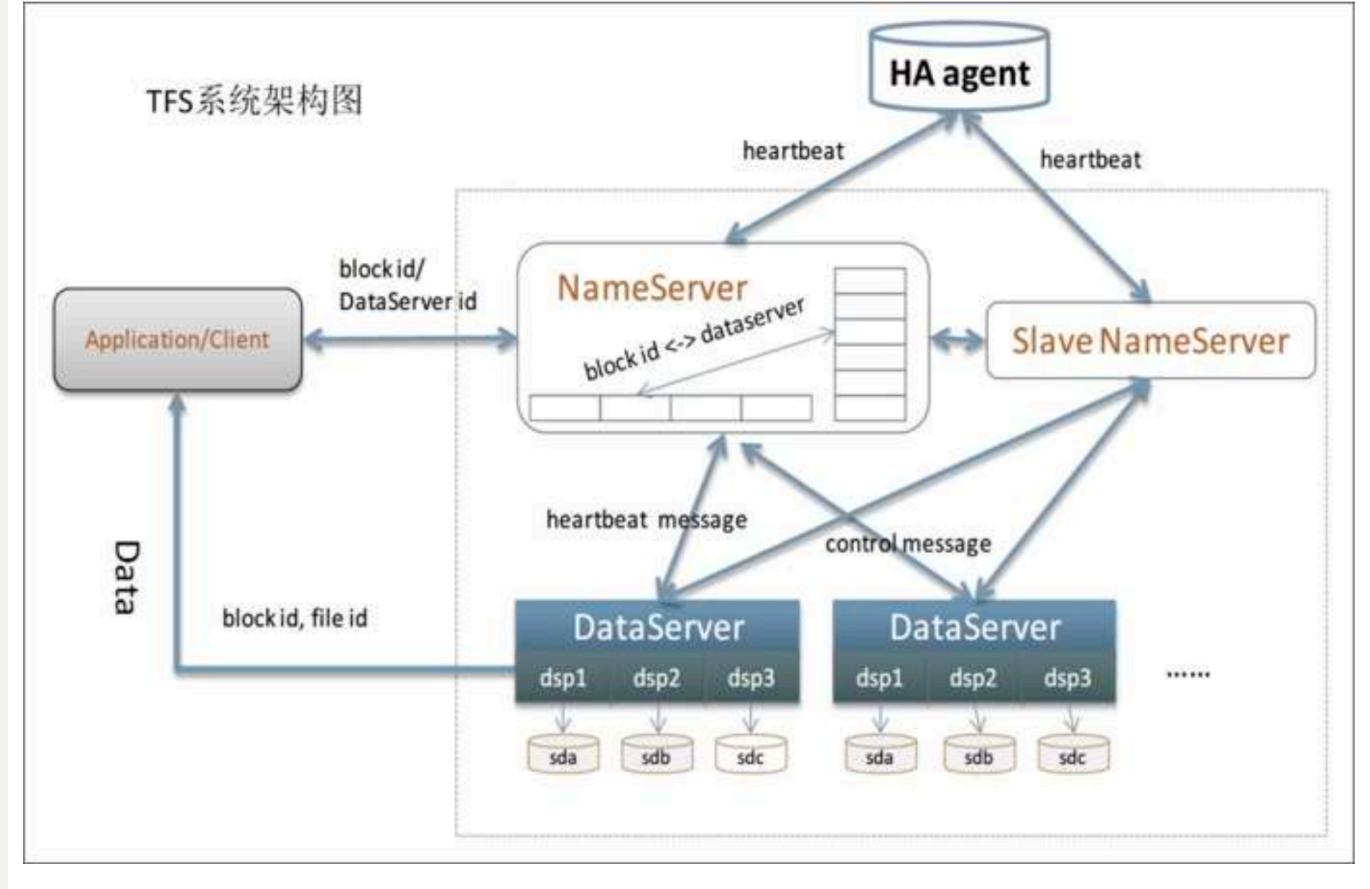
### 3、故障自动化处理

## 小文件存储

业务在起步阶段就可以做小文件统一存储，即使不是自研也要用个开源的。

典型的小文件存储有：淘宝的 TFS、京东 JFS、Facebook 的 Haystack。

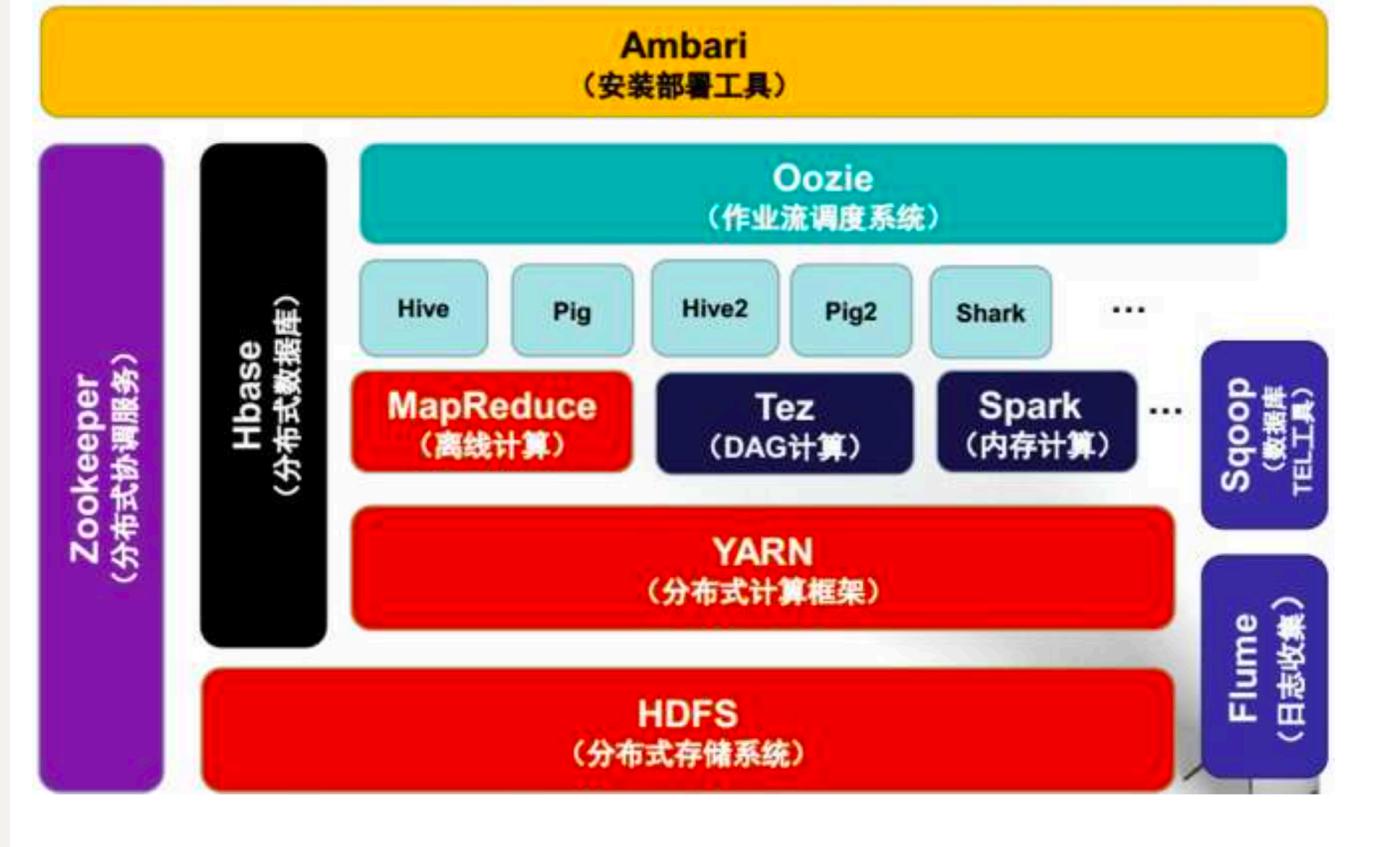
下图是淘宝 TFS 的架构：



## 大文件存储

互联网大文件主要分为两类：一类是业务数据，另一类是海量的日志数据。

下面是 Hadoop 的生态圈：



## 41. 互联网架构模板：“开发层”和“服务层”技术

### 开发层技术

#### 1、开发框架

互联网公司都会指定一个大的技术方向，然后使用统一的开发框架。这样可以避免技术人员之间没有共同语言，技术交流少；每类技术都需要投入大量的人力物力来学习和精通；不同团队之间的成员无法快速流动，人力资源不能高效利用。

对于框架选择有一个总的原则：优选成熟的框架，避免盲目追逐新技术。

#### 2、web服务器

独立开发一个成熟的web服务器，成本非常高，况且业界有那么多成熟的开源web服务器，所以互联网行业都是“拿来主义”，选择一款流行的开源框架即可。

#### 3、容器

Docker容易已经一统江湖。

## 服务层技术

服务层的主要目的就是为了降低系统间相互关联的复杂度。

1、配置中心

2、服务中心

3、消息队列

## 42、互联网架构模板：“网络层”技术

### 负载均衡

1、DNS

(1) .A记录 又称IP指向， 用户可以在此设置子域名并指向到自己的目标主机地址上， 从而实现通过域名找到服务器。说明：·指向的目标主机地址类型只能使用IP地址；

附加说明：

1) 泛域名解析即将该域名所有未指定的子域名都指向一个空间。在“主机名”中填入\*，“类型”为A，“IP地址/主机名”中填入web服务器的IP地址，点击“新增”按钮即可。

2) 负载均衡的实现：负载均衡(Server Load Balancing, SLB)是指在一系列资源上面动态地分布网络负载。负载均衡可以减少网络拥塞，提高整体网络性能，提高自愈性，并确保企业关键性应用的可用性。当相同子域名有多个目标地址时，表示轮循，可以达到负载均衡的目的，但需要虚拟主机服务商支持。

(2) .CNAME 通常称别名指向。您可以为一个主机设置别名。比如设置test.mydomain.com，用来指向一个主机www.rddns.com那么以后就可以用test.mydomain.com来代替访问www.rddns.com了。

说明：·

**CNAME**的目标主机地址只能使用主机名，不能使用IP地址；·主机名前不能有任何其他前缀，如：http://等是不被允许的；·A记录优先于CNAME记录。即如果一个主机地址同时存在A记录和CNAME记录，则CNAME记录不生效。

(3) .**MX记录** 邮件交换记录。用于将以该域名为结尾的电子邮件指向对应的邮件服务器以进行处理。如：用户所用的邮件是以域名mydomain.com为结尾的，则需要在管理界面中添加该域名的MX记录来处理所有以@mydomain.com结尾的邮件。

说明：·

**MX记录**可以使用主机名或IP地址；·**MX记录**可以通过设置优先级实现主辅服务器设置，“优先级”中的数字越小表示级别越高。也可以使用相同优先级达到负载均衡的目的；·如果在“主机名”中填入子域名则此MX记录只对该子域名生效。

附加说明：

1) 负载均衡服务器负载均衡(Server Load Balancing, SLB)是指在一系列资源上面智能地分布网络负载。负载均衡可以减少网络拥塞，提高整体网络性能，提高自愈性，并确保企业关键性应用的可用性。当域名的MX记录有多个目标地址且优先级相同时，表示轮循，可以达到负载均衡的目的，但需要邮箱服务商支持。

(4) .**NS记录** 解析服务器记录。用来表明由哪台服务器对该域名进行解析。这里的NS记录只对子域名生效。

例如用户希望由12.34.56.78这台服务器解析news.mydomain.com，则需要设置news.mydomain.com的NS记录。

说明：·“优先级”中的数字越小表示级别越高；·“IP地址/主机名”中既可以填写IP地址，也可以填写像ns.mydomain.com这样的主机地址，但必须保证该主机地址有效。

如，将news.mydomain.com的NS记录指向到ns.mydomain.com，在设置NS记录的同时还需要设置ns.mydomain.com的指向，

否则NS记录将无法正常解析；·NS记录优先于A记录。即，如果一个主机地址同时存在NS记录和A记录，则A记录不生效。这里的NS记录只对子域名生效。

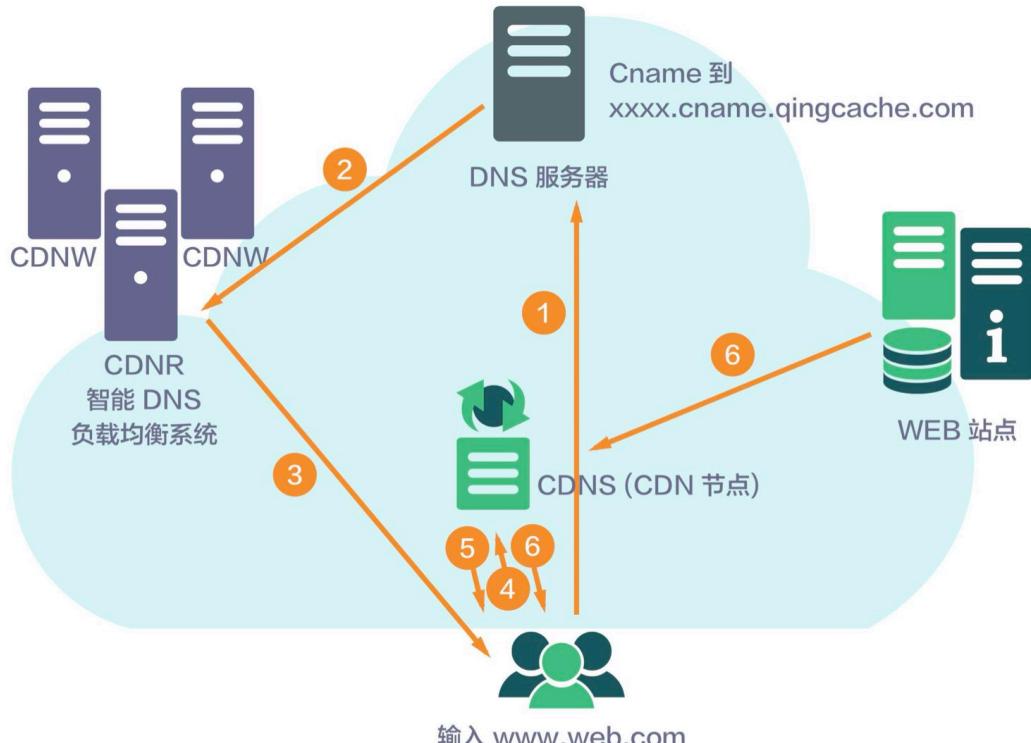
相关说明

1) 负载均衡服务器负载均衡(Server Load Balancing, SLB)是指在一系列资源上面智能地分布网络负载。负载均衡可以减少网络拥塞，提高整体网络性能，提高自愈性，并确保企业关键性应用的可用性。当相同子域有多个目标地址，或域名的MX记录有多个目标地址且优先级相同时，表示轮循，可以达到负载均衡的目的，但需要虚拟主机和邮箱服务商支持。

2) TTL值TTL值全称是“生存时间 (Time To Live)”，简单的说它表示DNS记录在DNS服务器上缓存时间。东方网景DNS服务器默认即时生效，客户的增加修改一般不超过15分钟可以使用。

## 2、Nginx、LVS、F5

## 3、CDN



## 多机房

### 1、同城多机房

同一个城市多个机房，距离不会太远，可以投入重金搭建私有的高速网络，基本上能够做到和同机房一样的想过。

2、跨城多机房

3、跨国多机房

多中心

部分特殊场景的业务的美好愿景?

多个机房都能提供服务，故障发生时自动切换。

## 43. 互联网架构模板：“用户层”和“业务层”技术

用户层技术

1、用户管理

用户众多，需要使用单点登录(SSO)，又叫统一登录。目前最为成熟的为CAS方案，架构为：<https://apereo.github.io/cas/4.2.x/planning/Architecture.html>

2、消息推送

消息推送根据不同的途径，分为短信、邮件、站内信和app推送。除了app，其他基本就是调用api即可完成，没有难度。

自己实现app推送的难点主要如下：

(1) 海量设备和用户管理

消息推送的设备数量众多，存储和管理这些设备是比较复杂的；同时为了对不同大用户进行不同的业务推广，还需要收集一些用户信息，简单来说就是将用户和设备关联起来，需要提取用户特征，为用户分类和达标签。

(2) 连接保活

也就是app在后台可能已经被关闭的情况下我们需要推送自己的消息，那么就会有很多黑科技的手法，比如应用相互拉起，找手机厂商开白名单等。

### (3) 消息管理

实际业务运营过程中，并不是每个消息都要发送给每个用户，而是可以根据用户的特征选择一些用户进行推送。由于用户特征变化很大，各种排列组合可能都会用，将消息推送给哪些用户是一个非常灵活的逻辑，所以功能设计也需要非常灵活才能有效的支持该业务。具体可以参考微内核。

## 3、存储云、图片云

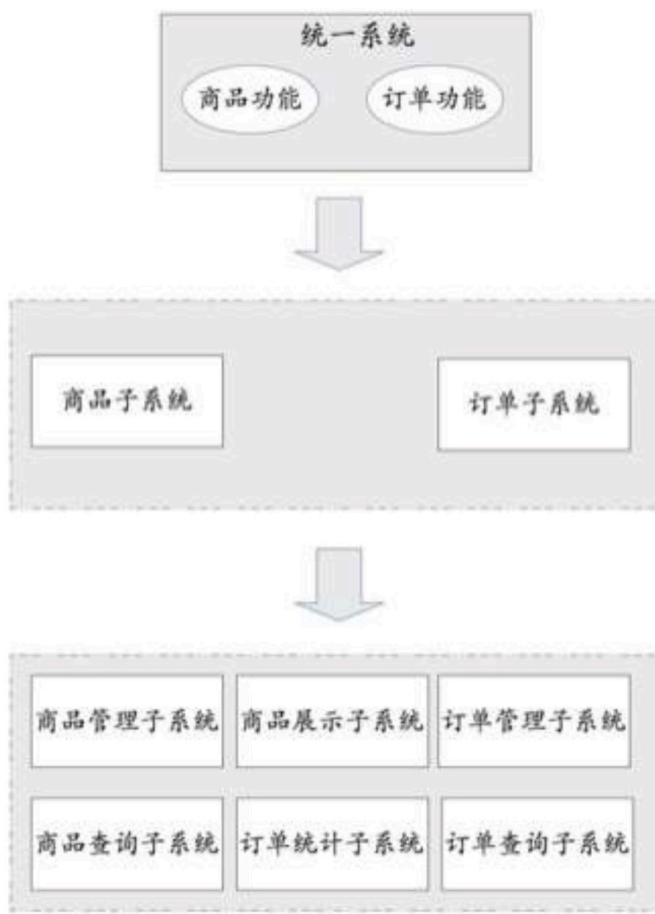
互联网业务场景中，用户会上传多种类型的数据文件，这些文件具备几个特点：

- (1) 数据量大：用户基数大，上传数据就越频繁
- (2) 文件体积小
- (3) 访问有时效性

文件和图片又有很大的区别，文件一般直接上传下载就可以了，但是图片会有很多复杂的操作，比如裁剪、缩放等。

## 业务层技术

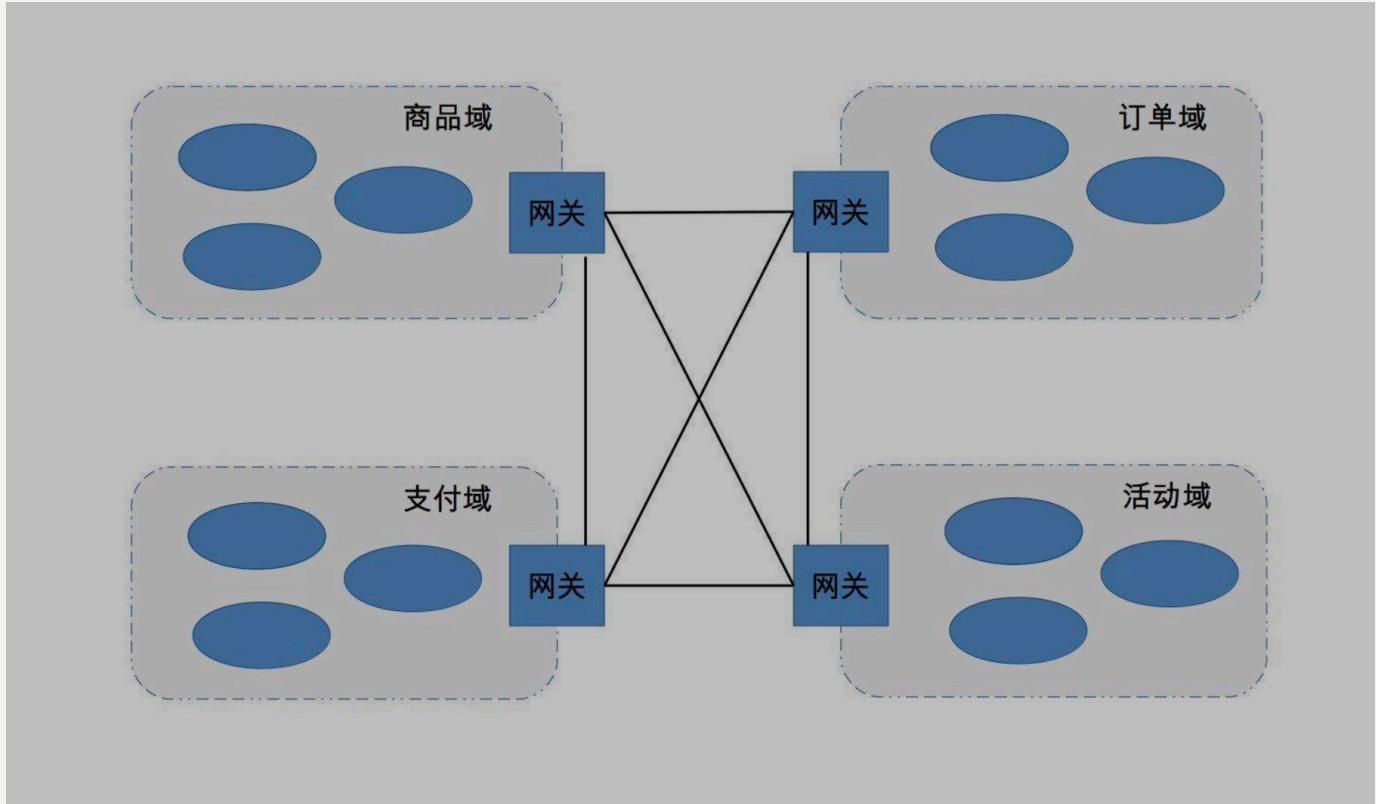
各个互联网企业面临的最终问题都是类似的：业务复杂度越来越高。也就是，业务层面临的主要技术挑战就是“复杂度高”。解决方法一个字“拆”。



我这个模拟的电商系统经历了 3 个发展阶段：

- 第一阶段：所有功能都在 1 个系统里面。
- 第二阶段：将商品和订单拆分到 2 个子系统里面。
- 第三阶段：商品子系统和订单子系统分别拆分成了更小的 6 个子系统。

随着子系统越来越多，如果达到几百甚至上千，另一个复杂问题就凸显出来了：子系统数量太多，已经没有人能说清楚业务的调用流程了，出了问题排查也会特比困难。所以这时候就是另外一种方式“合”，按照高内聚、低耦合的原则将职责关联比较强的子系统合成一个虚拟业务域，然后通过网关统一对外呈现，类似于设计模式中的facade模式。



## 44. 互联网架构模板：“平台技术”

### 运维平台

运维平台的核心职责分为四大块：“配置”、“部署”、“监控”、“应急”，每个职责对应系统生命周期的一个阶段。

运维平台的核心设计要素是“四化”：标准化、平台化、自动化、可视化。

#### 1、标准化

各系统都需要按照平台的标准来进行实现对接，然而如果有的系统确实无法满足平台的规范，那么应该由一个第三方来进行适配。

#### 2、平台化

将运维的相关操作都集成在平台中，通过运维平台来完成运维工作。

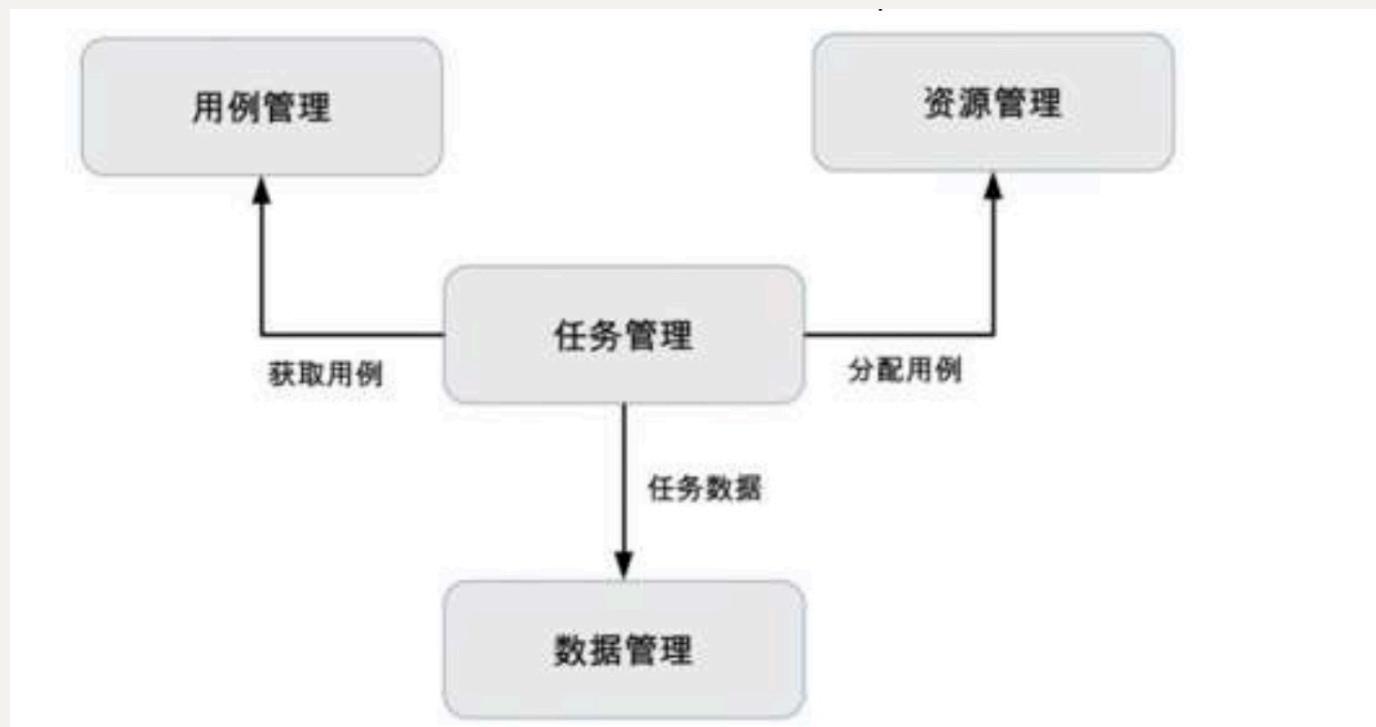
#### 3、自动化

传统手工运维效率低的主要原因就是存在大量的重复操作，运维平台可以将这些操作固化下来，由系统自动完成。

#### 4、可视化

### 测试平台

测试平台的功能包括单元测试、集成测试、接口测试、性能测试等。关键目标就是提升测试效率，从而提升产品质量，其设计关键就是自动化。



### 数据平台



## 管理平台

### 45. 架构重构内功心法第一式：有的放矢

相比于全新的架构来说，架构重构对于架构师的要求更高，主要体现在：

- (1) 业务已经上线，不能停下来。
- (2) 关联方众多，牵一发动全身。
- (3) 旧架构的约束。

在面临重构问题时，架构师的首要任务是从一大堆纷繁复杂的问题中识别出真正要通过架构来重构的问题，集中力量快速解决，而不是想着通过架构重构来解决所有问题。

一个简单的准则：假设我们现在需要从0开始设计当前系统，新架构和老架构是否类似？如果差异不大，说明系统采取优化即可；如果差异很大，那么就可能需要进行系统重构了。

## 46. 架构重构内功心法第二式：合纵连横

### 合纵

用通俗易懂的话语和相关的产品、测试、项目、运维进行有效沟通，将技术术语大白话后方便各方理解，认识到问题的存在和重构的必要性。

### 连横

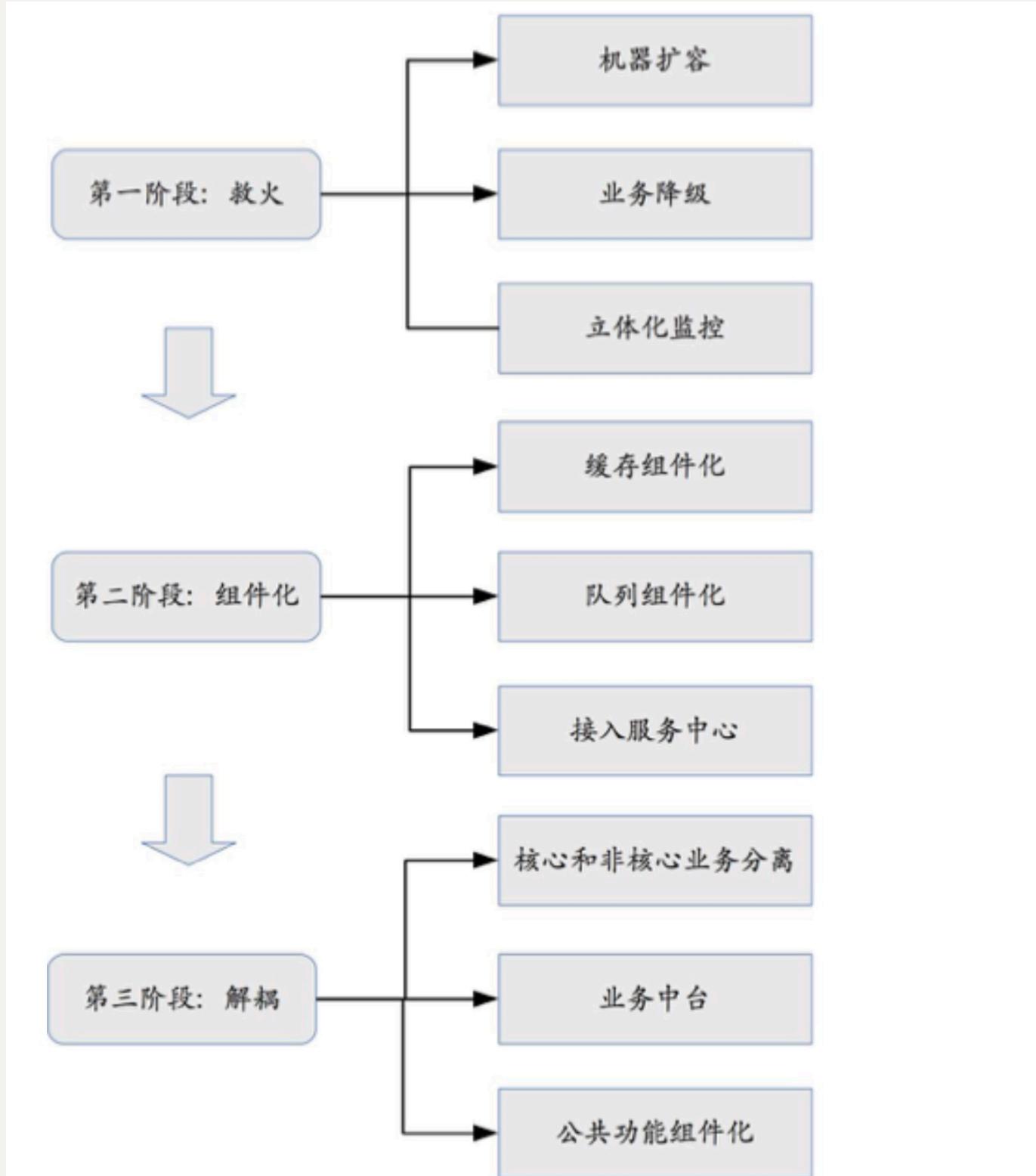
推动其他关联系统进行重构，这种人虽然技术术语很容易沟通，但是常常会面临两个问题：

(1) 这对我有什么好处

有效的策略”换位思考、合作双赢、关注长期“。简单来说就是站在对方的角度思考，重构对他有什么好处，解决什么问题，带来什么收益等。

(2) 这部分我这边现在不急

## 47. 架构重构内功心法第三式：运筹帷幄



重构的流程，其实就是“分阶段实施”，将要解决的问题根据优先级、重要性、实施难度等划分为不同的阶段，每个阶段聚焦与一个整体目标，集中精力和资源解决一类问题，这样做的好处有：

- (1) 每个阶段都有明确目标，做完之后效果明显，团队信心足，后续推进更加容易。
- (2) 每个阶段的工作量不会太大，可以和业务工作并行。

(3) 每个阶段的改动不会太大，降低了整体风险。

那么如何制定分阶段重构的策略呢？

(1) 优先级排序

将明显且比较紧急的事项有限落地，解决目前遇到的主要问题。

(2) 问题分类

将问题按照性质分类，每个解决集中解决一类问题。

(3) 先易后难

解决难的问题一般会依赖于先解决容易的问题；

最难的问题耗时耗资而且效果不明显，会打击士气造成后续活动的消极影响；

刚开始的分析不一定全面，所以一开始对于最难的最关键的问题可能会分析失败。

(4) 循序渐进

## 48.再谈开源项目：如何选择、使用以及二次开发

选：如何选择一个开源项目

1、聚焦是否满足业务

2、聚焦是否成熟

3、聚焦运维能力

## 用：如何使用开源项目

1、深入研究仔细测试

2、小心应用，灰度发布

3、做好应急，以防万一

## 改：如何基于开源项目做二次开发

1、保持纯洁，加以包装

2、发明你要的轮子

## 49.谈谈App架构的演进

1、WebAPP

2、原生APP

3、Hybrid APP

4、组件化和容器化

5、跨平台APP

## 50.架构设计文档模板

### 备选方案模板

1、需求介绍

**【需求介绍主要描述需求的背景、目标、范围等】**

2、需求分析

**【需求分析主要全方位地描述需求相关的信息】**

**【5W Who When What Where Why】**

**【Who: 利益干系人, 包括开发者、使用者、购买者、决策者等。】**

**【When: 需求使用时间, 包括季节、时间、里程碑等。】**

**【What: 需求产出的是什么, 包括系统、文件、开发库、数据、平台等】**

**【Where: 需求的应用场景, 包括国家、地点、环境等】**

**【Why: 需求需要解决的问题, 通常和需求背景相关】**

**【1H How】**

**【这里的How不是设计方案也不是架构方案, 而是关键业务流程】**

**【8C指的是8个约束和限制, 包含性能、成本、时间、可靠性、安全性、合规性、技术性、兼容性】**

### 3、复杂度分析

**【分析需求的复杂度, 复杂度常见的有高可用、高性能、可扩展等】**

### 4、备选方案

**【备选方案设计, 至少3个备选方案, 每个备选方案需要描述关键的实现, 无须描述具体的实现细节】**

### 5、评估备选方案

**【备选方案360环评表】**

# 架构设计模板

## 1、总体方案

【总体方案需要从整体上描述方案的结构，其核心就是架构图，以及针对架构图的描述，包括模块或者子系统的职责描述、核心流程】

## 2、架构总览

【架构总览给出架构图以及架构的描述】

## 3、核心流程

## 4、详细设计

## 5、架构演进规划

【通常就是各个阶段需要做什么，即第一期做什么，第二期做什么，第三期做什么】