针对大型网站的架构

架构体系——多系统间的通信问题，分布式协作

系统基础设施——保证系统基本运行

系统安全问题——防止非法操作

系统稳定性——保证系统正常运行

数据分析——提高业务质量

总结：希望业务能够安全、正常地发展，同时根据反馈改善自身业务，使其更好地发展，建立这样一个良性循环。

分类：

系统基础设施

架构体系

系统稳定性

系统安全问题

数据分析

架构体系

RPC

实现

TCP(Socket+ObjectStream字节流传输)

HTTP(web容器端口+string字符串传输，实质也是字节流传输)

具体实现

HTTPClient，servlet，springMVC

优势：容易理解，通用性强，线程问题开箱即用

劣势：协议上层——效率低

格式标准：

JSON jackson-all

XML：Extensible Markup Language xtream-1.4.4.jar

URL风格：

RESTFul /xx/provider/sayHelloService/xxaaaaaa.json

RPC xx.do?service=xx.yy.zz&method=fff

包含概念

HTTP服务网关 gateway

路由

负载均衡

分类

硬件：F5

软件：LVS（传输层），Nginx（应用层）

算法

轮询法，随机法，原地址hash法，加权轮询，加权随机，最小连接

动态配置

脚本语言——Groovy

Zookeeper

集群：leader election

结点：persistent，ephemeral

watcher：触发一次

facade：

zkClient

问题：

单点

解决

服务单机->双机 一台stand by，Zk注册服务地址

序列化

传输

时延性

A同B异

A同B同

A异B同

A异B异

系统基础设施

缓存 解决传统数据库磁盘低吞吐

memcache

实现：libevent

算法：

LRU

api：

java：Memcahed-Java-Client

特点：

非分布式->自行hash实现->导致雪崩效应->改善：consistent hash

分布式session

coockie 大小受限，安全性差

持久化到DB

持久化到缓存集群

实现：

tomcat实现：memcached-session-manager，Sticky和Non-Sticky模式

数据库

需求理论

CAP

ACID

类型：

传统：

MySQL(复杂查询、事务)

扩展：

业务拆分：DB->MultiDB

复制策略：Master-Slave，Dual-Master

分表:Order->Order\_1..Order\_n 解决量大查询，但不解决写

分表分库：db.Order->db\_0.Order\_1,db\_n.Order\_y

海量DB:

HBase（难支持复杂查询，但写性能高，易扩展）

概念:

原型Google BigTable,运行于HDFS

集群角色：HMaster和HRegionServer

行、列族、列、版本（时间戳）

迭代行

rowkey->

rowkey's range []

全表扫描 开销大

优化：

rowkey按商品id\_时间生成

二级索引表，存在性能、一致性问题

Key—Value

Redis:(多种存储结构，集群，吞吐量，并发数，支持MQ)

api：

java：Jedis

消息

JMS规范

消息类型：

发送类型：

实现

ActiveMQ:

搜索引擎

其它

1. a
2. fsdfds
3. sdfsfs
   1. dsfsdf
   2. fsdf

三层架构。。UI (表现层) BLL（[业务逻辑层](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91%E5%B1%82&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuh79PW6YmW6dPvNWn1Dk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nsPjR4PWD3)） DAL（[数据访问层](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E8%AE%BF%E9%97%AE%E5%B1%82&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuh79PW6YmW6dPvNWn1Dk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nsPjR4PWD3)）

饿了么



**WebAPI**主要做一些 HTTPS 卸载、限流，还有安全校验等一些通用的和业务逻辑无关的操作。

**Service Orchestrator** 是服务编排层，通过配置的方式实现内外网的协议转换、服务的聚合裁剪。

右边是一些围绕这些服务化框架的辅助系统，比如说用于定期执行一个任务的 Job 系统。

服务拆分

**把一些基础服务抽象出来。**

比如说支付业务，业务很单一的时候，紧耦合的代码没有关系，但是扩展出越来越多业务都需要支付服务的时候，你每一个业务（比如说支付的功能）都要去做一个吗？所以我们要把这些基础服务抽离出来。**比如说支付服务、短信服务、推送服务等。**

拆服务看似很简单、没什么价值，但这恰恰是我们刚开始就要做的事情。其实在这个时期，前面所有的那些架构都可以往后拖，因为不做架构调整其实不会死人，但是拆服务你不做的话，真的会死人的。

服务拆分必定是一个漫长的过程，这实际上是一个很痛苦的过程，也是需要很多配套系统的系统工程。

发布系统

**发布是最大的不稳定因素。**

**发布的最大问题在于发布上去之后没有简单可执行的回退操作。**

所以我们就需**要有发布系统，发布系统定义了统一的回退操作，所有服务必须遵循发布系统的定义回退操作。**

服务框架

**分布式服务框架包含的服务注册、发现、负载均衡、路由、流控、熔断、降级等功能**

**这对我们后来一些中间件的选型是有影响的，比如说 DAL 层。**

DAL 数据访问层

**做 DAL 服务层的时候，第一件事情是限流，**其他的东西还可以放一放。**然后做连接复用**

**多进程之间其实是不可以共享一个连接的。**

**我们 DAL 层要做一个连接复用。**

这个连接复用讲的不是服务本身的连接复用，而是说 DAL 层上的连接复用，就是服务有 1000 个连接到 DAL 层，经过连接复用后对数据库可能只是保持着十几个连接。一旦发现某个数据库请求是一个事务的话，那么 DAL 就帮你保留这个连接的对应关系。当这个事务结束之后，就把数据库的连接，放回到共用池里面去，供其他人使用。

**然后做冒烟和熔断。**数据库也可以熔断的。当数据库发生冒烟时，我们会杀掉一些数据库的请求，保证数据库不至于崩溃。

服务治理

**服务框架之后，涉及服务治理的问题。**

**首先是埋点，**你要埋很多很多的监控点。

我们的资源和时间总是有限的，**作为架构师和 CTO 来说，如何在这种有限的资源下，产出更重要的东西？**

流控系统

升级方式