Unknown date | jason9263

jason9263/architect-awesome: 后端架构师技术图谱

《后端架构师技术图谱》

更新于20180513

        - 堆排序
        - 计数排序
        - 桶排序
        - 基数排序
        - 二分查找
        - Java 中的排序工具
    - 布隆过滤器
    - 字符串比较
        - KMP 算法
    - 深度优先、广度优先
    - 贪心算法
    - 回溯算法
    - 剪枝算法
    - 动态规划
    - 朴素贝叶斯
    - 推荐算法
    - 最小生成树算法
    - 最短路径算法
- 并发
    - 多线程
    - 线程安全
    - 一致性、事务
        - 事务 ACID 特性
        - 事务的隔离级别
        - MVCC
    - 锁
        - Java中的锁和同步类
        - 公平锁 & 非公平锁
        - 悲观锁
        - 乐观锁 & CAS
        - ABA 问题
        - CopyOnWrite容器
        - RingBuffer
        - 可重入锁 & 不可重入锁
        - 互斥锁 & 共享锁
        - 死锁
- 操作系统
    - 计算机原理
    - CPU
        - 多级缓存
    - 进程
    - 线程

    - 协程
    - Linux
- 设计模式
    - 设计模式的六大原则
    - 23种常见设计模式
    - 应用场景
    - 单例模式
    - 责任链模式
    - MVC
    - IOC
    - AOP
    - UML
    - 微服务思想
        - 康威定律
- 运维 & 统计 & 技术支持
    - 常规监控
    - APM
    - 统计分析
    - 持续集成(CI/CD)
        - Jenkins
        - 环境分离
    - 自动化运维
        - Ansible
        - puppet
        - chef
    - 测试
        - TDD 理论
        - 单元测试
        - 压力测试
        - 全链路压测
        - A/B 、灰度、蓝绿测试
    - 虚拟化
        - KVM
        - Xen
        - OpenVZ
    - 容器技术
        - Docker
    - 云技术
        - OpenStack
    - DevOps
    - 文档管理
- 中间件

    - Web Server
        - Nginx
        - OpenResty
        - Apache Httpd
        - Tomcat
            - 架构原理
            - 调优方案
        - Jetty
    - 缓存
        - 本地缓存
    - 客户端缓存
    - 服务端缓存
        - Web缓存
        - Memcached
        - Redis
            - 架构
            - 回收策略
        - Tair
    - 消息队列
        - 消息总线
        - 消息的顺序
        - RabbitMQ
        - RocketMQ
        - ActiveMQ
        - Kafka
        - Redis 消息推送
        - ZeroMQ
    - 定时调度
        - 单机定时调度
        - 分布式定时调度
    - RPC
        - Dubbo
        - Thrift
        - gRPC
    - 数据库中间件
        - Sharding Jdbc
    - 日志系统
        - 日志搜集
    - 配置中心
    - API 网关
- 网络
    - 协议

        - OSI 七层协议
        - TCP/IP
        - HTTP
        - HTTP2.0
        - HTTPS
    - 网络模型
        - Epoll
        - Java NIO
        - kqueue
    - 连接和短连接
    - 框架
    - 零拷贝（Zero-copy）
    - 序列化(二进制协议)
        - Hessian
        - Protobuf
- 数据库
    - 基础理论
        - 数据库设计的三大范式
    - MySQL
        - 原理
        - InnoDB
        - 优化
        - 索引
            - 聚集索引, 非聚集索引
            - 复合索引
            - 自适应哈希索引(AHI)
        - explain
    - NoSQL
        - MongoDB
        - Hbase
- 搜索引擎
    - 搜索引擎原理
    - Lucene
    - Elasticsearch
    - Solr
    - sphinx
- 性能
    - 性能优化方法论
    - 容量评估
    - CDN 网络
    - 连接池
    - 性能调优

- 大数据
  - 流式计算
    - Storm
    - Flink
    - Kafka Stream
    - 应用场景
  - Hadoop
    - HDFS
    - MapReduce
    - Yarn
  - Spark
- 安全
  - web 安全
    - XSS
    - CSRF
    - SQL 注入
    - Hash Dos
    - 脚本注入
    - 漏洞扫描工具
    - 验证码
  - DDoS 防范
  - 用户隐私信息保护
  - 序列化漏洞
  - 加密解密
    - 对称加密
    - 哈希算法
    - 非对称加密
  - 服务器安全
  - 数据安全
    - 数据备份
  - 网络隔离
    - 内外网分离
    - 登录跳板机
  - 授权、认证
    - RBAC
    - OAuth2.0
    - 双因素认证（2FA）
    - 单点登录(SSO)
- 常用开源框架
  - 开源协议
  - 日志框架
    - Log4j、Log4j2

数据结构

队列

- 《java队列——queue详细分析》
    - 非阻塞队列：ConcurrentLinkedQueue(无界线程安全)，采用CAS机制（compareAndSwapObject原子操作）。
    - 阻塞队列：ArrayBlockingQueue(有界)、LinkedBlockingQueue（无界）、DelayQueue、PriorityBlockingQueue，采用锁机制；使用ReentrantLock 锁。
- 《LinkedList、ConcurrentLinkedQueue、LinkedBlockingQueue对比分析》

集合

- 《Java Set集合的详解》

链表、数组

- 《Java集合详解--什么是List》

字典、关联数组

- 《Java map 详解 - 用法、遍历、排序、常用API等》

栈

- 《java数据结构与算法之栈（Stack）设计与实现》
- 《Java Stack 类》
- 《java stack的详细实现分析》
  - Stack 是线程安全的。
  - 内部使用数组保存数据，不够时翻倍。

树

二叉树

每个节点最多有两个叶子节点。

- 《二叉树》

完全二叉树

- 《完全二叉树》
  - 叶节点只能出现在最下层和次下层，并且最下面一层的结点都集中在该层最左边的若干位置的二叉树。

平衡二叉树

左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。

- 《浅谈数据结构-平衡二叉树》
- 《浅谈算法和数据结构: 八 平衡查找树之2-3树》

二叉查找树（BST）

二叉查找树（Binary Search Tree），也称有序二叉树（ordered binary tree）,排序二叉树（sorted binary tree）。

- 《浅谈算法和数据结构: 七 二叉查找树》

红黑树

- 《最容易懂得红黑树》
  - 添加阶段后，左旋或者右旋从而再次达到平衡。
- 《浅谈算法和数据结构: 九 平衡查找树之红黑树》

B-，B+，B*树

MySQL是基于B+树聚集索引组织表

- 《B-树，B+树，B*树详解》
- 《B-树，B+树与B*树的优缺点比较》
  - B+ 树的叶子节点链表结构相比于 B- 树便于扫库，和范围检索。

LSM 树

LSM（Log-Structured Merge-Trees）和 B+ 树相比，是牺牲了部分读的性能来换取写的性能(通过批量写入)，实现读写之间的。 Hbase、LevelDB、Tair（Long DB）、nessDB 采用 LSM 树的结构。LSM可以快速建立索引。

- 《LSM树 VS B+树》
  - B+ 树读性能好，但由于需要有序结构，当key比较分散时，磁盘寻道频繁，造成写性能。
  - LSM 是将一个大树拆分成N棵小树，先写到内存（无寻道问题，性能高），在内存中构建一颗有序小树（有序树），随着小树越来越大，内存的小树会flush到磁盘上。当读时，由于不知道数据在哪棵小树上，因此必须遍历（二分查找）所有的小树，但在每颗小树内部数据是有序的。
- 《LSM树（Log-Structured Merge Tree）存储引擎》
  - 极端的说，基于LSM树实现的HBase的写性能比MySQL高了一个数量级，读性能低了一个数量级。
  - 优化方式：Bloom filter 替代二分查找；compact 小数位大树，提高查询性能。
  - Hbase 中，内存中达到一定阈值后，整体flush到磁盘上、形成一个文件（B+数），HDFS不支持update操作，所以Hbase做整体flush而不是merge update。flush到磁盘上的小树，定期会合并成一个大树。

BitSet

经常用于大规模数据的排重检查。

- 《Java Bitset类》
- 《Java BitSet（位集）》

常用算法

- 《常见排序算法及对应的时间复杂度和空间复杂度》

排序、查找算法

- 《常见排序算法及对应的时间复杂度和空间复杂度》

选择排序

- 《Java中的经典算法之选择排序（SelectionSort）》
  - 每一趟从待排序的记录中选出最小的元素，顺序放在已排好序的序列最后，直到全部记录排序完毕。

冒泡排序

- 《冒泡排序的2种写法》
  - 相邻元素前后交换、把最大的排到最后。
  - 时间复杂度 $O(n^2)$

插入排序

- 《排序算法总结之插入排序》

快速排序

- 《坐在马桶上看算法：快速排序》
  - 一侧比另外一次都大或小。

归并排序

- 《图解排序算法(四)之归并排序》
  - 分而治之，分成小份排序，在合并(重建一个新空间进行复制)。

希尔排序

TODO

堆排序

- 《图解排序算法(三)之堆排序》
  - 排序过程就是构建最大堆的过程，最大堆：每个结点的值都大于或等于其左右孩子结点的值，堆顶元素是最大值。

计数排序

- 《计数排序和桶排序》
  - 和桶排序过程比较像，差别在于桶的数量。

桶排序

- 《【啊哈！算法】最快最简单的排序——桶排序》
- 《排序算法（三）：计数排序与桶排序》
  - 桶排序将[0,1)区间划分为n个相同的大小的子区间，这些子区间被称为桶。
  - 每个桶单独进行排序，然后再遍历每个桶。

基数排序

按照个位、十位、百位、...依次来排。

- 《排序算法系列：基数排序》
- 《基数排序》

二分查找

- 《二分查找(java实现)》

    - 要求待查找的序列有序。
    - 时间复杂度 O(logN)。

- 《java实现二分查找-两种方式》
    - while + 递归。

Java 中的排序工具

- 《Arrays.sort和Collections.sort实现原理解析》
    - Collections.sort算法调用的是合并排序。
    - Arrays.sort() 采用了2种排序算法 -- 基本类型数据使用快速排序法，对象数组使用归并排序。

布隆过滤器

常用于大数据的排重，比如email，url 等。 核心原理：将每条数据通过计算产生一个指纹（一个字节或多个字节，但一定比原始数据要少很多），其中每一位都是通过随机计算获得，在将指纹映射到一个大的按位存储的空间中。注意：会有一定的错误率。 优点：空间和时间效率都很高。 缺点：随着存入的元素数量增加，误算率随之增加。

- 《布隆过滤器 -- 空间效率很高的数据结构》
- 《大量数据去重：Bitmap和布隆过滤器(Bloom Filter)》
- 《基于Redis的布隆过滤器的实现》
    - 基于 Redis 的 Bitmap 数据结构。
- 《网络爬虫：URL去重策略之布隆过滤器(BloomFilter)的使用》
    - 使用Java中的 BitSet 类 和 加权和hash算法。

字符串比较

KMP 算法

KMP：Knuth-Morris-Pratt算法（简称KMP） 核心原理是利用一个“部分匹配表”，跳过已经匹配过的元素。

- 《字符串匹配的KMP算法》

深度优先、广度优先

- 《广度优先搜索BFS和深度优先搜索DFS》

贪心算法

- 《算法：贪婪算法基础》
- 《常见算法及问题场景——贪心算法》

回溯算法

- 《 五大常用算法之四：回溯法》

剪枝算法

- 《α－β剪枝算法》

动态规划

- 《详解动态规划——邹博讲动态规划》
- 《动态规划算法的个人理解》

朴素贝叶斯

- 《带你搞懂朴素贝叶斯分类算法》
    - P(B|A)=P(A|B)P(B)/P(A)
- 《贝叶斯推断及其互联网应用1》
- 《贝叶斯推断及其互联网应用2》

推荐算法

- 《推荐算法综述》
- 《TOP 10 开源的推荐系统简介》

最小生成树算法

- 《算法导论--最小生成树（Kruskal和Prim算法）》

最短路径算法

- 《Dijkstra算法详解》

并发

Java 并发

- Java 并发知识合集
- JAVA并发知识图谱

多线程

- 《40个Java多线程问题总结》

线程安全

- 《Java并发编程——线程安全及解决机制简介》

一致性、事务

事务 ACID 特性

- 《数据库事务ACID特性》

事务的隔离级别

- 未提交读：一个事务可以读取另一个未提交的数据，容易出现脏读的情况。
- 读提交：一个事务等另外一个事务提交之后才可以读取数据，但会出现不可重复读的情况（多次读取的数据不一致），读取过程中出现UPDATE操作，会多。（大多数数据库默认级别是RC，比如SQL Server，Oracle），读取的时候不可以修改。
- 可重复读： 同一个事务里确保每次读取的时候，获得的是同样的数据，但不保障原始数据被其他事务更新（幻读），Mysql InnoDB 就是这个级别。
- 序列化：所有事物串行处理（牺牲了效率）
- 《理解事务的4种隔离级别》
- 数据库事务的四大特性及事务隔离级别
- 《MySQL的InnoDB的幻读问题 》
  - 幻读的例子非常清楚。
  - 通过 SELECT ... FOR UPDATE 解决。
- 《一篇文章带你读懂MySQL和InnoDB》
  - 图解脏读、不可重复读、幻读问题。

MVCC

- 《【mysql】关于innodb中MVCC的一些理解》
  - innodb 中 MVCC 用在 Repeatable-Read 隔离级别。
  - MVCC 会产生幻读问题（更新时异常。）
- 《轻松理解MYSQL MVCC 实现机制》
  - 通过隐藏版本列来实现 MVCC 控制，一列记录创建时间、一列记录删除时间，这里的时间
  - 每次只操作比当前版本小（或等于）的 行。

锁

Java中的锁和同步类

- 《Java中的锁分类》
  - 主要包括 synchronized、ReentrantLock、和 ReadWriteLock。
- 《Java并发之AQS详解》
- 《Java中信号量 Semaphore》
  - 有数量控制
  - 申请用 acquire，申请不要则阻塞；释放用 release。
- 《java开发中的Mutex vs Semaphore》
  - 简单的说 就是Mutex是排它的，只有一个可以获取到资源， Semaphore也具有排它性，但可以定义多个可以获取的资源的对象。

公平锁 & 非公平锁

公平锁的作用就是严格按照线程启动的顺序来执行的，不允许其他线程插队执行的；而非公平锁是允许插队的。

- 《公平锁与非公平锁》
  - 默认情况下 ReentrantLock 和 synchronized 都是非公平锁。ReentrantLock 可以设置成公平锁。

悲观锁

悲观锁如果使用不当（锁的条数过多），会引起服务大面积等待。推荐优先使用乐观锁+重试。

- 《【MySQL】悲观锁&乐观锁》
  - 乐观锁的方式：版本号+重试方式
  - 悲观锁：通过 select ... for update 进行行锁(不可读、不可写，share 锁可读不可写)。
- 《Mysql查询语句使用select.. for update导致的数据库死锁分析》
  - mysql的innodb存储引擎实务锁虽然是锁行，但它内部是锁索引的。
  - 锁相同数据的不同索引条件可能会引起死锁。
- 《Mysql并发时经典常见的死锁原因及解决方法》

乐观锁 & CAS

- 《乐观锁的一种实现方式——CAS》
  - 和MySQL乐观锁方式相似，只不过是通过和原值进行比较。

ABA 问题

由于高并发，在CAS下，更新后可能此A非彼A。通过版本号可以解决，类似于上文 Mysql 中提到的的乐观锁。

- 《Java CAS 和ABA问题》
- 《Java 中 ABA问题及避免》
  - AtomicStampedReference 和 AtomicStampedReference。

CopyOnWrite容器

可以对CopyOnWrite容器进行并发的读，而不需要加锁。CopyOnWrite并发容器用于读多写少的并发场景。比如白名单，黑名单，商品类目的访问和更新场景，不适合需要数据强一致性的场景。

- 《JAVA中写时复制(Copy-On-Write)Map实现》

    - 实现读写分离，读取发生在原始数据上，写入发生在副本上。
    - 不用加锁，通过最终一致实现一致性。

- 《聊聊并发-Java中的Copy-On-Write容器》

RingBuffer

- 《线程安全的无锁RingBuffer的实现【一个读线程，一个写线程】》

可重入锁 & 不可重入锁

- 《可重入锁和不可重入锁》

    - 通过简单代码举例说明可重入锁和不可重入锁。
    - 可重入锁指同一个线程可以再次获得之前已经获得的锁。
    - 可重入锁可以用户避免死锁。
    - Java中的可重入锁：synchronized 和 java.util.concurrent.locks.ReentrantLock

- 《ReenTrantLock可重入锁（和synchronized的区别）总结》

    - synchronized 使用方便，编译器来加锁，是非公平锁。
    - ReenTrantLock 使用灵活，锁的公平性可以定制。
    - 相同加锁场景下，推荐使用 synchronized。

互斥锁 & 共享锁

互斥锁：同时只能有一个线程获得锁。比如，ReentrantLock 是互斥锁，ReadWriteLock 中的写锁是互斥锁。 共享锁：可以有多个线程同时或的锁。比如，Semaphore、CountDownLatch 是共享锁，ReadWriteLock 中的读锁是共享锁。

- 《ReadWriteLock场景应用》

死锁

- 《“死锁”四个必要条件的合理解释》

    - 互斥、持有、不可剥夺、环形等待。

- Java如何查看死锁？

    - JConsole 可以识别死锁。

- java多线程系列：死锁及检测
    - jstack 可以显示死锁。

操作系统

计算机原理

- 《操作系统基础知识——操作系统的原理，类型和结构》

CPU

多级缓存

典型的 CPU 有三级缓存，距离核心越近，速度越快，空间越小。L1 一般 32k，L2 一般 256k，L3 一般12M。内存速度需要200个 CPU 周期，CPU 缓存需要1个CPU周期。

- 《从Java视角理解CPU缓存和伪共享》

进程

TODO

线程

- 《线程的生命周期及状态转换详解》

协程

- 《终结python协程----从yield到actor模型的实现》
    - 线程的调度是由操作系统负责，协程调度是程序自行负责
    - 与线程相比，协程减少了无谓的操作系统切换.
    - 实际上当遇到IO操作时做切换才更有意义，（因为IO操作不用占用CPU），如果没遇到IO操作，按照时间片切换.

Linux

- 《Linux 命令大全》

设计模式

设计模式的六大原则

- 《设计模式的六大原则》
  - 开闭原则：对扩展开放,对修改关闭，多使用抽象类和接口。
  - 里氏替换原则：基类可以被子类替换，使用抽象类继承,不使用具体类继承。
  - 依赖倒转原则：要依赖于抽象,不要依赖于具体，针对接口编程,不针对实现编程。
  - 接口隔离原则：使用多个隔离的接口,比使用单个接口好，建立最小的接口。
  - 迪米特法则：一个软件实体应当尽可能少地与其他实体发生相互作用，通过中间类建立联系。
  - 合成复用原则：尽量使用合成/聚合,而不是使用继承。

23种常见设计模式

- 《设计模式》
- 《23种设计模式全解析》

应用场景

- 《细数JDK里的设计模式》
  - 结构型模式：
    - 适配器：用来把一个接口转化成另一个接口，如 java.util.Arrays#asList()。
    - 桥接模式：这个模式将抽象和抽象操作的实现进行了解耦，这样使得抽象和实现可以独立地变化，如JDBC；
    - 组合模式：使得客户端看来单个对象和对象的组合是同等的。换句话说，某个类型的方法同时也接受自身类型作为参数，如 Map.putAll，List.addAll、Set.addAll。
    - 装饰者模式：动态的给一个对象附加额外的功能，这也是子类的一种替代方式，如 java.util.Collections#checkedList|Map|Set|SortedSet|SortedMap。
    - 享元模式：使用缓存来加速大量小对象的访问时间，如 valueOf(int)。
    - 代理模式：代理模式是用一个简单的对象来代替一个复杂的或者创建耗时的对象，如 java.lang.reflect.Proxy

    - 创建模式:
        - 抽象工厂模式：抽象工厂模式提供了一个协议来生成一系列的相关或者独立的对象，而不用指定具体对象的类型，如java.util.Calendar#getInstance()。
        - 建造模式(Builder)：定义了一个新的类来构建另一个类的实例，以简化复杂对象的创建，如：java.lang.StringBuilder#append()。
        - 工厂方法：就是 一个返* 回具体对象的方法，而不是多个，如java.lang.Object#toString()、java.lang.Class#newInstance()。
        - 原型模式：使得类的实例能够生成自身的拷贝、如：java.lang.Object#clone()。
        - 单例模式：全局只有一个实例，如java.lang.Runtime#getRuntime()。
    - 行为模式：
        - 责任链模式：通过把请求从一个对象传递到链条中下一个对象的方式，直到请求被处理完毕，以实现对象间的解耦。如javax.servlet.Filter#doFilter()。
        - 命令模式：将操作封装到对象内，以便存储，传递和返回，如：java.lang.Runnable。
        - 解释器模式：定义了一个语言的语法，然后解析相应语法的语句，如，java.text.Format，java.text.Normalizer。
        - 迭代器模式：提供一个一致的方法来顺序访问集合中的对象，如java.util.Iterator。
        - 中介者模式：通过使用一个中间对象来进行消息分发以及减少类之间的直接依赖，java.lang.reflect.Method#invoke()。
        - 空对象模式：如 java.util.Collections#emptyList()。
        - 观察者模式：它使得一个对象可以灵活的将消息发送给感兴趣的对象，如 java.util.EventListener。
        - 模板方法模式：让子类可以重写方法的一部分，而不是整个重写，如java.util.Collections#sort()。

- 《Spring-涉及到的设计模式汇总》
- 《Mybatis使用的设计模式》

单例模式

- 《单例模式的三种实现 以及各自的优缺点》
- 《单例模式－－反射－－防止序列化破坏单例模式》
    - 使用枚举类型。

责任链模式

TODO

MVC

- 《MVC 模式》
  - 模型(model)－视图(view)－控制器(controller)

IOC

- 《理解 IOC》
- 《IOC 的理解与解释》
  - 正向控制：传统通过new的方式。反向控制，通过容器注入对象。
  - 作用：用于模块解耦。
  - DI：Dependency Injection，即依赖注入，只关心资源使用，不关心资源来源。

AOP

- 《轻松理解AOP(面向切面编程)》
- 《Spring AOP详解》
- 《Spring AOP的实现原理》
  - Spring AOP使用的动态代理，主要有两种方式：JDK动态代理和CGLIB动态代理。
- 《Spring AOP 实现原理与 CGLIB 应用》
  - Spring AOP 框架对 AOP 代理类的处理原则是：如果目标对象的实现类实现了接口，Spring AOP 将会采用 JDK 动态代理来生成 AOP 代理类；如果目标对象的实现类没有实现接口，Spring AOP 将会采用 CGLIB 来生成 AOP 代理类

UML

- 《UML教程》

微服务思想

- 《微服务架构设计》
- 《微服务架构技术栈选型手册》

康威定律

- 《微服务架构的理论基础 - 康威定律》

    - 定律一：组织沟通方式会通过系统设计表达出来，就是说架构的布局和组织结构会有相似。
    - 定律二：时间再多一件事情也不可能做的完美，但总有时间做完一件事情。一口气吃不成胖子，先搞定能搞定的。
    - 定律三：线型系统和线型组织架构间有潜在的异质同态特性。种瓜得瓜，做独立自治的子系统减少沟通成本。
    - 定律四：大的系统组织总是比小系统更倾向于分解。合久必分，分而治之。

- 《微服务架构核心20讲》

运维 & 统计 & 技术支持

常规监控

- 《腾讯业务系统监控的修炼之路》
    - 监控的方式：主动、被动、旁路(比如舆情监控)
    - 监控类型： 基础监控、服务端监控、客户端监控、 监控、用户端监控
    - 监控的目标：全、块、准
    - 核心指标：请求量、成功率、耗时
- 《开源还是商用？十大云运维监控工具横评》
    - Zabbix、Nagios、Ganglia、Zenoss、Open-falcon、监控宝、 360网站服务监控、阿里云监控、百度云观测、小蜜蜂网站监测等。
- 《监控报警系统搭建及二次开发经验》

命令行监控工具

- 《常用命令行监控工具》
    - top、sar、tsar、nload
- 《20个命令行工具监控 Linux 系统性能》
- 《JVM性能调优监控工具jps、jstack、jmap、jhat、jstat、hprof使用详解》

APM

APM — Application Performance Management

- 《Dapper，大规模分布式系统的跟踪系统》

- CNCF OpenTracing，中文版
- 主要开源软件，按字母排序
  - Apache SkyWalking
  - CAT
  - CNCF jaeger
  - Pinpoint
  - Zipkin
- 《开源APM技术选型与实战》
  - 主要基于 Google的Dapper（大规模分布式系统的跟踪系统） 思想。

统计分析

- 《流量统计的基础：埋点》
  - 常用指标：访问与访客、停留时长、跳出率、退出率、转化率、参与度
- 《APP埋点常用的统计工具、埋点目标和埋点内容》
  - 第三方统计：友盟、百度移动、魔方、App Annie、talking data、神策数据等。
- 《美团点评前端无痕埋点实践》
  - 所谓无痕、即通过可视化工具配置采集节点，在前端自动解析配置并上报埋点数据，而非硬编码。

持续集成(CI/CD)

- 《持续集成是什么？》
- 《8个流行的持续集成工具》

Jenkins

- 《使用Jenkins进行持续集成》

环境分离

开发、测试、生成环境分离。

- 《开发环境、生产环境、测试环境的基本理解和区》

自动化运维

Ansible

- 《Ansible中文权威指南》
- 《Ansible基础配置和企业级项目实用案例》

puppet

- 《自动化运维工具——puppet详解》

chef

- 《Chef 的安装与使用》

测试

TDD 理论

- 《深度解读 - TDD（测试驱动开发）》
  - 基于测试用例编码功能代码，XP（Extreme Programming）的核心实践.
  - 好处：一次关注一个点，降低思维负担；迎接需求变化或改善代码的设计；提前澄清需求；快速反馈；

单元测试

- 《Java单元测试之JUnit篇》
- 《JUnit 4 与 TestNG 对比》
  - TestNG 覆盖 JUnit 功能，适用于更复杂的场景。
- 《单元测试主要的测试功能点》
  - 模块接口测试、局部数据结构测试、路径测试 、错误处理测试、边界条件测试 。

压力测试

- 《Apache ab 测试使用指南》

- 《大型网站压力测试及优化方案》
- 《10大主流压力/负载/性能测试工具推荐》
- 《真实流量压测工具 tcpcopy应用浅析》
- 《nGrinder 简易使用教程》

全链路压测

- 《京东618：升级全链路压测方案，打造军演机器人ForceBot》
- 《饿了么全链路压测的探索与实践》
- 《四大语言，八大框架丨滴滴全链路压测解决之道》
- 《全链路压测经验》

A/B 、灰度、蓝绿测试

- 《技术干货丨AB 测试和灰度发布探索及实践》
- 《nginx 根据IP 进行灰度发布》
- 《蓝绿部署、A/B 测试以及灰度发布》

虚拟化

- 《VPS的三种虚拟技术OpenVZ、Xen、KVM优缺点比较》

KVM

- 《KVM详解，太详细太深入了，经典》
- 《【图文】KVM 虚拟机安装详解》

Xen

- 《Xen虚拟化基本原理详解》

OpenVZ

- 《开源Linux容器 OpenVZ 快速上手指南》

容器技术

Docker

- 《几张图帮你理解 docker 基本原理及快速入门》
- 《Docker 核心技术与实现原理》
- 《Docker 教程》

云技术

OpenStack

- 《OpenStack构架知识梳理》

DevOps

- 《一分钟告诉你究竟DevOps是什么鬼？》
- 《DevOps详解》

文档管理

- Confluence-收费文档管理系统
- GitLab?
- Wiki

中间件

Web Server

Nginx

- 《Ngnix的基本学习-多进程和Apache的比较》
    - Nginx 通过异步非阻塞的事件处理机制实现高并发。Apache 每个请求独占一个线程，非常消耗系统资源。
    - 事件驱动适合于IO密集型服务(Nginx)，多进程或线程适合于CPU密集型服务(Apache)，所以Nginx适合做反向代理，而非web服务器使用。
- 《nginx与Apache的对比以及优缺点》
    - nginx只适合静态和反向代理，不适合处理动态请求。

OpenResty

- 官方网站
- 《浅谈 OpenResty》
  - 通过 Lua 模块可以在Nginx上进行开发。

Apache Httpd

- 官方网站

Tomcat

架构原理

- 《TOMCAT原理详解及请求过程》
- 《Tomcat服务器原理详解》
- 《Tomcat 系统架构与设计模式,第 1 部分: 工作原理》
- 《四张图带你了解Tomcat系统架构》
- 《JBoss vs. Tomcat: Choosing A Java Application Server》
  - Tomcat 是轻量级的 Serverlet 容器，没有实现全部 JEE 特性（比如持久化和事务处理），但可以通过其他组件代替，比如Srping。
  - Jboss 实现全部了JEE特性，软件开源免费、文档收费。

调优方案

- 《Tomcat 调优方案》
  - 启动NIO模式（或者APR）；调整线程池；禁用AJP连接器（Nginx+tomcat的架构，不需要AJP）；
- 《tomcat http协议与ajp协议》
- 《AJP与HTTP比较和分析》
  - AJP 协议（8009端口）用于降低和前端Server（如Apache，而且需要支持AJP协议）的连接数(前端)，通过长连接提高性能。
  - 并发高时，AJP协议优于HTTP协议。

Jetty

- 《Jetty 的工作原理以及与 Tomcat 的比较》
- 《jetty和tomcat优势比较》
  - 架构比较:Jetty的架构比Tomcat的更为简单。
  - 性能比较：Jetty和Tomcat性能方面差异不大，Jetty默认采用NIO结束在处理I/O请求上更占优势，Tomcat默认采用BIO处理I/O请求，Tomcat适合处理少数非常繁忙的链接，处理静态资源时性能较差。
  - 其他方面：Jetty的应用更加快速，修改简单，对新的Servlet规范的支持较好;Tomcat 对JEE和Servlet 支持更加全面。

缓存

- 《缓存失效策略（FIFO 、LRU、LFU三种算法的区别）》

本地缓存

- 《HashMap本地缓存》
- 《EhCache本地缓存》
  - 堆内、堆外、磁盘三级缓存。
  - 可按照缓存空间容量进行设置。
  - 按照时间、次数等过期策略。
- 《Guava Cache》
  - 简单轻量、无堆外、磁盘缓存。
- 《Nginx本地缓存》
- 《Pagespeed—懒人工具，服务器端加速》

客户端缓存

- 《浏览器端缓存》
  - 主要是利用 Cache-Control 参数。
- 《H5 和移动端 WebView 缓存机制解析与实战》

服务端缓存

Web缓存

- nuster - nuster cache
- varnish - varnish cache
- squid - squid cache

Memcached

- 《Memcached 教程》
- 《深入理解Memcached原理》
  - 采用多路复用技术提高并发性。
  - slab分配算法： memcached给Slab分配内存空间，默认是1MB。分配给Slab之后 把slab的切分成大小相同的chunk，Chunk是用于缓存记录的内存空间，Chunk 的大小默认按照1.25倍的速度递增。好处是不会频繁申请内存，提高IO效率，坏处是会有一定的内存浪费。
- 《Memcached软件工作原理》
- 《Memcache技术分享：介绍、使用、存储、算法、优化、命中率》
- 《memcache 中 add 、 set 、replace 的区别》
  - 区别在于当key存在还是不存在时，返回值是true和false的。
- 《memcached全面剖析》

Redis

- 《Redis 教程》
- 《redis底层原理》
  - 使用 ziplist 存储链表，ziplist是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中的。
  - 使用 skiplist(跳跃表)来存储有序集合对象、查找上先从高Level查起、时间复杂度和红黑树相当，实现容易，无锁、并发性好。
- 《Redis持久化方式》
  - RDB方式：定期备份快照，常用于灾难恢复。优点：通过fork出的进程进行备份，不影响主进程、RDB 在恢复大数据集时的速度比 AOF 的恢复速

度要快。缺点：会丢数据。
    - AOF方式：保存操作日志方式。优点：恢复时数据丢失少，缺点：文件大，回复慢。
    - 也可以两者结合使用。

- 《分布式缓存--序列3--原子操作与CAS乐观锁》

架构

- 《Redis单线程架构》

回收策略

- 《redis的回收策略》

Tair

- 官方网站
- 《Tair和Redis的对比》
- 特点：可以配置备份节点数目，通过异步同步到备份节点
- 一致性Hash算法。
- 架构：和Hadoop 的设计思想类似，有Configserver，DataServer，Configserver 通过心跳来检测，Configserver也有主备关系。

几种存储引擎:

- MDB，完全内存性，可以用来存储Session等数据。
- Rdb（类似于Redis），轻量化，去除了aof之类的操作，支持Restfull操作
- LDB（LevelDB存储引擎），持久化存储，LDB 作为rdb的持久化，google实现，比较高效，理论基础是LSM(Log-Structured-Merge Tree)算法，现在内存中修改数据，达到一定量时（和内存汇总的旧数据一同写入磁盘）再写入磁盘，存储更加高效，县比喻Hash算法。
- Tair采用共享内存来存储数据，如果服务挂掉（非服务器），重启服务之后，数据亦然还在。

消息队列

- 《消息队列-推/拉模式学习 & ActiveMQ及JMS学习》
    - RabbitMQ 消费者默认是推模式（也支持拉模式）。
    - Kafka 默认是拉模式。
    - Push方式：优点是可以尽可能快地将消息发送给消费者，缺点是如果消费者处理能力跟不上，消费者的缓冲区可能会溢出。

    - Pull方式：优点是消费端可以按处理能力进行拉去，缺点是会增加消息延迟。

- 《Kafka、RabbitMQ、RocketMQ等消息中间件的对比 —— 消息发送性能和区别》

消息总线

消息总线相当于在消息队列之上做了一层封装，统一入口，统一管控、简化接入成本。

- 《消息总线VS消息队列》

消息的顺序

- 《如何保证消费者接收消息的顺序》

RabbitMQ

支持事务，推拉模式都是支持、适合需要可靠性消息传输的场景。

- 《RabbitMQ的应用场景以及基本原理介绍》
- 《消息队列之 RabbitMQ》
- 《RabbitMQ之消息确认机制（事务+Confirm）》

RocketMQ

Java实现，推拉模式都是支持，吞吐量逊于Kafka。可以保证消息顺序。

- 《RocketMQ 实战之快速入门》
- 《RocketMQ 源码解析》

ActiveMQ

纯Java实现，兼容JMS，可以内嵌于Java应用中。

- 《ActiveMQ消息队列介绍》

Kafka

高吞吐量、采用拉模式。适合高IO场景，比如日志同步。

- 官方网站
- 《各消息队列对比，Kafka深度解析，众人推荐，精彩好文！》
- 《Kafka分区机制介绍与示例》

Redis 消息推送

生产者、消费者模式完全是客户端行为，list 和 拉模式实现，阻塞等待采用 blpop 指令。

- 《Redis学习笔记之十：Redis用作消息队列》

ZeroMQ

TODO

定时调度

单机定时调度

- 《linux定时任务cron配置》
- 《Linux cron运行原理》
    - fork 进程 + sleep 轮询
- 《Quartz使用总结》
- 《Quartz源码解析 ---- 触发器按时启动原理》
- 《quartz原理揭秘和源码解读》
    - 定时调度在 QuartzSchedulerThread 代码中，while()无限循环，每次循环取出时间将到的trigger，触发对应的job，直到调度器线程被关闭。

分布式定时调度

- 《这些优秀的国产分布式任务调度系统，你用过几个？》
    - opencron、LTS、XXL-JOB、Elastic-Job、Uncode-Schedule、Antares
- 《Quartz任务调度的基本实现原理》
    - Quartz集群中，独立的Quartz节点并不与另一其的节点或是管理节点通信，而是通过相同的数据库表来感知到另一Quartz应用的
- 《Elastic-Job-Lite 源码解析》
- 《Elastic-Job-Cloud 源码解析》

RPC

- 《从零开始实现RPC框架 - RPC原理及实现》
    - 核心角色：Server: 暴露服务的服务提供方、Client: 调用远程服务的服务消费方、Registry: 服务注册与发现的注册中心。
- 《分布式RPC框架性能大比拼 dubbo、motan、rpcx、gRPC、thrift的性能比较》

Dubbo

- 官方网站
- dubbo实现原理简单介绍

** SPI ** TODO

Thrift

- 官方网站
- 《Thrift RPC详解》
    - 支持多语言，通过中间语言定义接口。

---

gRPC

服务端可以认证加密，在外网环境下，可以保证数据安全。

- 官方网站

- 《你应该知道的RPC原理》

数据库中间件

Sharding Jdbc

- 官网

日志系统

日志搜集

- 《从零开始搭建一个ELKB日志收集系统》
- 《用ELK搭建简单的日志收集分析系统》
- 《日志收集系统-探究》

配置中心

- Apollo - 携程开源的配置中心应用
    - Spring Boot 和 Spring Cloud
    - 支持推、拉模式更新配置
    - 支持多种语言
- 《基于zookeeper实现统一配置管理》
- 《 Spring Cloud Config 分布式配置中心使用教程》

servlet 3.0 异步特性可用于配置中心的客户端

- 《servlet3.0 新特性——异步处理》

API 网关

主要职责：请求转发、安全认证、协议转换、容灾。

- 《API网关那些儿》
- 《谈API网关的背景、架构以及落地方案》
- 《使用Zuul构建API Gateway》
- 《Spring Cloud Gateway 源码解析》

- 《HTTP API网关选择之一Kong介绍》

网络

协议

OSI 七层协议

- 《OSI七层协议模型、TCP/IP四层模型学习笔记》

TCP/IP

- 《深入浅出 TCP/IP 协议》
- 《TCP协议中的三次握手和四次挥手》

HTTP

- 《http协议详解(超详细)》

HTTP2.0

- 《HTTP 2.0 原理详细分析》
- 《HTTP2.0的基本单位为二进制帧》
  - 利用二进制帧负责传输。
  - 多路复用。

HTTPS

- 《https原理通俗了解》
  - 使用非对称加密协商加密算法
  - 使用对称加密方式传输数据
  - 使用第三方机构签发的证书，来加密公钥，用于公钥的安全传输、防止被中间人串改。
- 《八大免费SSL证书-给你的网站免费添加Https安全加密》

网络模型

- 《web优化必须了解的原理之I/o的五种模型和web的三种工作模式》
  - 五种I/O模型：阻塞I/O，非阻塞I/O，I/O复用、事件(信号)驱动I/O、异步I/O，前四种I/O属于同步操作，I/O的第一阶段不同、第二阶段相同，最后的一种则属于异步操作。
  - 三种 Web Server 工作方式：Prefork(多进程)、Worker方式(线程方式)、Event方式。
- 《select、poll、epoll之间的区别总结》
  - select，poll，epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的。
  - select 有打开文件描述符数量限制，默认1024（2048 for x64），100万并发，就要用1000个进程、切换开销大；poll采用链表结构，没有数量限制。
  - select，poll “醒着”的时候要遍历整个fd集合，而epoll在“醒着”的时候只要判断一下就绪链表是否为空就行了，通过回调机制节省大量CPU时间；select，poll每次调用都要把fd集合从用户态往内核态拷贝一次，而epoll只要一次拷贝。
  - poll会随着并发增加，性能逐渐下降，epoll采用红黑树结构，性能稳定，不会随着连接数增加而降低。
- 《select，poll，epoll比较 》
  - 在连接数少并且连接都十分活跃的情况下，select和poll的性能可能比epoll好，毕竟epoll的通知机制需要很多函数回调。
- 《深入理解Java NIO》
  - NIO 是一种同步非阻塞的 IO 模型。同步是指线程不断轮询 IO 事件是否就绪，非阻塞是指线程在等待 IO 的时候，可以同时做其他任务
- 《BIO与NIO、AIO的区别》
- 《两种高效的服务器设计模型：Reactor和Proactor模型》

Epoll

- 《epoll使用详解（精髓）》

Java NIO

- 《深入理解Java NIO》
- 《Java NIO编写Socket服务器的一个例子》

---

kqueue

- 《kqueue用法简介》

连接和短连接

- 《TCP/IP系列——长连接与短连接的区别》

框架

- 《Netty原理剖析》
  - Reactor 模式介绍。
  - Netty 是 Reactor 模式的一种实现。

零拷贝（Zero-copy）

- 《对于 Netty ByteBuf 的零拷贝(Zero Copy) 的理解》
  - 多个物理分离的buffer，通过逻辑上合并成为一个，从而避免了数据在内存之间的拷贝。

序列化(二进制协议)

Hessian

- 《Hessian原理分析》 Binary-RPC;不仅仅是序列化

Protobuf

- 《Protobuf协议的Java应用例子》 Goolge出品、占用空间和效率完胜其他序列化类库，如Hessian；需要编写 .proto 文件。
- 《Protocol Buffers序列化协议及应用》
  - 关于协议的解释；缺点：可读性差;
- 《简单的使用 protobuf 和 protostuff》
  - protostuff 的好处是不用写 .proto 文件，Java 对象直接就可以序列化。

数据库

基础理论

数据库设计的三大范式

- 《数据库的三大范式以及五大约束》
  - 第一范式：数据表中的每一列（每个字段）必须是不可拆分的最小单元，也就是确保每一列的原子性；
  - 第二范式（2NF）：满足1NF后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情；
  - 第三范式：必须先满足第二范式（2NF），要求：表中的每一列只与主键直接相关而不是间接相关，（表中的每一列只能依赖于主键）；

MySQL

原理

- 《MySQL的InnoDB索引原理详解》
- 《MySQL存储引擎——MyISAM与InnoDB区别》
  - 两种类型最主要的差别就是Innodb 支持事务处理与外键和行级锁
- 《myisam和innodb索引实现的不同》

InnoDB

- 《一篇文章带你读懂Mysql和InnoDB》

优化

- 《MySQL36条军规》
- 《MYSQL性能优化的最佳20+条经验》
- 《SQL优化之道》

- 《mysql数据库死锁的产生原因及解决办法》
- 《导致索引失效的可能情况》
- 《 MYSQL分页limit速度太慢优化方法》
    - 原则上就是缩小扫描范围。

索引

聚集索引, 非聚集索引

- 《MySQL 聚集索引/非聚集索引简述》
- 《MyISAM和InnoDB的索引实现》

MyISAM 是非聚集，InnoDB 是聚集

复合索引

- 《复合索引的优点和注意事项》

自适应哈希索引(AHI)

- 《InnoDB存储引擎——自适应哈希索引》

---

explain

- 《MySQL 性能优化神器 Explain 使用分析》

NoSQL

MongoDB

- MongoDB 教程
- 《Mongodb相对于关系型数据库的优缺点》
    - 优点：弱一致性（最终一致），更能保证用户的访问速度；内置GridFS，支持大容量的存储；Schema-less 数据库，不用预先定义结构；内置Sharding；相比于其他NoSQL，第三方支持丰富；性能优越；
    - 缺点：mongodb不支持事务操作；mongodb占用空间过大；MongoDB没有如MySQL那样成熟的维护工具，这对于开发和IT运营都是个值得注意的地方；

Hbase

- 《简明 HBase 入门教程（开篇）》
- 《深入学习HBase架构原理》
- 《传统的行存储和（HBase）列存储的区别》
- 《Hbase与传统数据库的区别》
    - 空数据不存储，节省空间，且适用于并发。
- 《HBase Rowkey设计》
    - rowkey 按照字典顺序排列，便于批量扫描。
    - 通过散列可以避免热点。

搜索引擎

搜索引擎原理

- 《倒排索引--搜索引擎入门》

Lucene

- 《Lucene入门简介》

Elasticsearch

- 《Elasticsearch学习，请先看这一篇！》
- 《Elasticsearch索引原理》

Solr

- 《 Apache Solr入门教程》
- 《elasticsearch与solr比较》

sphinx

- 《Sphinx 的介绍和原理探索》

性能

性能优化方法论

- 《15天的性能优化工作，5方面的调优经验》
    - 代码层面、业务层面、数据库层面、服务器层面、前端优化。
- 《系统性能优化的几个方面》

容量评估

- 《联网性能与容量评估的方法论和典型案例》
- 《互联网架构，如何进行容量设计？》
    - 评估总访问量、评估平均访问量QPS、评估高峰QPS、评估系统、单机极限QPS

CDN 网络

- 《CDN加速原理》
- 《国内有哪些比较好的 CDN？》

连接池

- 《主流Java数据库连接池比较与开发配置实战》

性能调优

- 《九大Java性能调试工具，必备至少一款》

大数据

流式计算

Storm

- 官方网站
- 《最详细的Storm入门教程》

Flink

- 《Flink之一 Flink基本原理介绍》

Kafka Stream

- 《Kafka Stream调研：一种轻量级流计算模式》

应用场景

例如：

- 广告相关实时统计；
- 推荐系统用户画像标签实时更新；
- 线上服务健康状况实时监测；
- 实时榜单；
- 实时数据统计。

Hadoop

- 《用通俗易懂的话说下hadoop是什么,能做什么》
- 《史上最详细的Hadoop环境搭建》

HDFS

- 《【Hadoop学习】HDFS基本原理》

MapReduce

- 《用通俗易懂的大白话讲解Map/Reduce原理》
- 《 简单的map-reduce的java例子》

Yarn

- 《初步掌握Yarn的架构及原理》

Spark

- 《Spark(一): 基本架构及原理》

安全

web 安全

XSS

- 《xss攻击原理与解决方法》

CSRF

- 《CSRF原理及防范》

SQL 注入

- 《SQL注入》

Hash Dos

- 《邪恶的JAVA HASH DOS攻击》
  - 利用JsonObjet 上传大Json，JsonObject 底层使用HashMap；不同的数据产生相同的hash值，使得构建Hash速度变慢，耗尽CPU。
- 《一种高级的DoS攻击-Hash碰撞攻击》
- 《关于Hash Collision DoS漏洞：解析与解决方案》

脚本注入

- 《上传文件漏洞原理及防范》

漏洞扫描工具

- 《DVWA》
- W3af
- OpenVAS详解

验证码

- 《验证码原理分析及实现》
- 《详解滑动验证码的实现原理》
    - 滑动验证码是根据人在滑动滑块的响应时间，拖拽速度，时间，位置，轨迹，重试次数等来评估风险。
- 《淘宝滑动验证码研究》

DDoS 防范

- 《学习手册：DDoS的攻击方式及防御手段》
- 《免费DDoS攻击测试工具大合集》

用户隐私信息保护

1. 用户密码非明文保存，加动态salt。
2. 身份证号，手机号如果要显示，用“*”替代部分字符。
3. 联系方式在的显示与否由用户自己控制。
4. TODO

- 《个人隐私包括哪些》
- 《在互联网上，隐私的范围包括哪些？》
- 《用户密码保存》

序列化漏洞

- 《Lib之过？Java反序列化漏洞通用利用分析》

加密解密

对称加密

- 《常见对称加密算法》
    - DES、3DES、Blowfish、AES
    - DES 采用 56位秘钥，Blowfish 采用1到448位变长秘钥，AES 128，192和256位长度的秘钥。
    - DES 秘钥太短（只有56位）算法目前已经被 AES 取代，并且 AES 有硬件加速，性能很好。

哈希算法

- 《常用的哈希算法》
  - MD5 和 SHA-1 已经不再安全，已被弃用。
  - 目前 SHA-256 是比较安全的。
- 《基于Hash摘要签名的公网URL签名验证设计方案》

非对称加密

- 《常见非对称加密算法》
  - RSA、DSA、ECDSA(螺旋曲线加密算法)
  - 和 RSA 不同的是 DSA 仅能用于数字签名，不能进行数据加密解密，其安全性和RSA相当，但其性能要比RSA快。
  - 256位的ECC秘钥的安全性等同于3072位的RSA秘钥。

    《区块链的加密技术》

服务器安全

- 《Linux强化论：15步打造一个安全的Linux服务器》

数据安全

数据备份

TODO

网络隔离

内外网分离

TODO

登录跳板机

在内外环境中通过跳板机登录到线上主机。

- 《搭建简易堡垒机》

授权、认证

RBAC

- 《基于组织角色的权限设计》
- 《权限系统与RBAC模型概述》
- 《Spring整合Shiro做权限控制模块详细案例分析》

OAuth2.0

- 《理解OAuth 2.0》
- 《一张图搞定OAuth2.0》

双因素认证（2FA）

2FA - Two-factor authentication，用于加强登录验证

常用做法是 登录密码 + 手机验证码（或者令牌Key，类似于与网银的 USB key）

- 【《双因素认证（2FA）教程》】(http://www.ruanyifeng.com/blog/2017/11/2fa-tutorial.html)

单点登录(SSO)

- 《单点登录原理与简单实现》
- CAS单点登录框架

常用开源框架

开源协议

- 《开源协议的选择》
- 如何选择一个开源软件协议

日志框架

Log4j、Log4j2

- 《log4j 详细讲解》
- 《log4j2 实际使用详解》
- 《Log4j1,Logback以及Log4j2性能测试对比》
    - Log4J 异步日志性能优异。

Logback

- 《最全LogBack 详解、含java案例和配置说明》

ORM

- 《ORM框架使用优缺点》
    - 主要目的是为了提高开发效率。

MyBatis：

- 《mybatis缓存机制详解》
    - 一级缓存是SqlSession级别的缓存，缓存的数据只在SqlSession内有效
    - 二级缓存是mapper级别的缓存，同一个namespace公用这一个缓存，所以对SqlSession是共享的；使用 LRU 机制清理缓存，通过 cacheEnabled 参数开启。
- 《MyBatis学习之代码生成器Generator》

网络框架

TODO

Web 框架

Spring 家族

Spring

- Spring 简明教程

Spring Boot

- 官方网站
- 《Spring Boot基础教程》

Spring Cloud

- Spring Boot 中文索引站
- Spring Cloud 中文文档
- 《Spring Cloud基础教程》

工具框架

- 《Apache Commons 工具类介绍及简单使用》
- 《Google guava 中文教程》

分布式设计

扩展性设计

- 《架构师不可不知的十大可扩展架构》
    - 总结下来，通用的套路就是分布、缓存及异步处理。
- 《可扩展性设计之数据切分》
    - 水平切分+垂直切分
    - 利用中间件进行分片如，MySQL Proxy。
    - 利用分片策略进行切分，如按照ID取模。
- 《说说如何实现可扩展性的大型网站架构》
    - 分布式服务+消息队列。
- 《大型网站技术架构（七）--网站的可扩展性架构》

稳定性 & 高可用

- 《系统设计：关于高可用系统的一些技术方案》
    - 可扩展：水平扩展、垂直扩展。 通过冗余部署，避免单点故障。
    - 隔离：避免单一业务占用全部资源。避免业务之间的相互影响 2. 机房隔离避免单点故障。
    - 解耦：降低维护成本，降低耦合风险。减少依赖，减少相互间的影响。

    - 限流：滑动窗口计数法、漏桶算法、令牌桶算法等算法。遇到突发流量时，保证系统稳定。
    - 降级：紧急情况下释放非核心功能的资源。牺牲非核心业务，保证核心业务的高可用。
    - 熔断：异常情况超出阈值进入熔断状态，快速失败。减少不稳定的外部依赖对核心服务的影响。
    - 自动化测试：通过完善的测试，减少发布引起的故障。
    - 灰度发布：灰度发布是速度与安全性作为妥协，能够有效减少发布故障。

- 《关于高可用的系统》

    - 设计原则：数据不丢(持久化)；服务高可用(服务副本)；绝对的100%高可用很难，目标是做到尽可能多的9，如99.999%（全年累计只有5分钟）。

硬件负载均衡

- 《转！！负载均衡器技术Nginx和F5的优缺点对比》

    - 主要是和F5对比。

- 《软/硬件负载均衡产品 你知多少？》

软件负载均衡

- 《几种负载均衡算法》 轮寻、权重、负载、最少连接、QoS
- 《DNS负载均衡》

    - 配置简单，更新速度慢。

- 《Nginx负载均衡》

    - 简单轻量、学习成本低；主要适用于web应用。

- 《借助LVS+Keepalived实现负载均衡 》

    - 配置比较负载、只支持到4层，性能较高。

- 《HAProxy用法详解 全网最详细中文文档》

    - 支持到七层（比如HTTP）、功能比较全面，性能也不错。

- 《Haproxy+Keepalived+MySQL实现读均衡负载》

    - 主要是用户读请求的负载均衡。

- 《rabbitmq+haproxy+keepalived实现高可用集群搭建》

限流

- 《谈谈高并发系统的限流》
    - 计数器：通过滑动窗口计数器，控制单位时间内的请求次数，简单粗暴。
    - 漏桶算法：固定容量的漏桶，漏桶满了就丢弃请求，比较常用。
    - 令牌桶算法：固定容量的令牌桶，按照一定速率添加令牌，处理请求前需要拿到令牌，拿不到令牌则丢弃请求，或进入丢队列，可以通过控制添加令牌的速率，来控制整体速度。Guava 中的 RateLimiter 是令牌桶的实现。
    - Nginx 限流：通过 `limit_req` 等模块限制并发连接数。

应用层容灾

- 《防雪崩利器：熔断器 Hystrix 的原理与使用》
    - 雪崩效应原因：硬件故障、硬件故障、程序Bug、重试加大流量、用户大量请求。
    - 雪崩的对策：限流、改进缓存模式(缓存预加载、同步调用改异步)、自动扩容、降级。
    - Hystrix设计原则：
        - 资源隔离：Hystrix通过将每个依赖服务分配独立的线程池进行资源隔离, 从而避免服务雪崩。
        - 熔断开关：服务的健康状况 = 请求失败数 / 请求总数，通过阈值设定和滑动窗口控制开关。
        - 命令模式：通过继承 HystrixCommand 来包装服务调用逻辑。

- 《缓存穿透，缓存击穿，缓存雪崩解决方案分析》

- 《缓存击穿、失效以及热点key问题》
    - 主要策略：失效瞬间：单机使用锁；使用分布式锁；不过期；
    - 热点数据：热点数据单独存储；使用本地缓存；分成多个子key；

跨机房容灾

- 《“异地多活”多机房部署经验谈》

    - 通过自研中间件进行数据同步。

- 《异地多活（异地双活）实践经验》
    - 注意延迟问题，多次跨机房调用会将延时放大数倍。
    - 建房间专线很大概率会出现问题，做好运维和程序层面的容错。
    - 不能依赖于程序端数据双写，要有自动同步方案。
    - 数据永不在高延迟和较差网络质量下，考虑同步质量问题。
    - 核心业务和次要业务分而治之，甚至只考虑核心业务。
    - 异地多活监控部署、测试也要跟上。
    - 业务允许的情况下考虑用户分区，尤其是游戏、邮箱业务。
    - 控制跨机房消息体大小，越小越好。
    - 考虑使用docker容器虚拟化技术，提高动态调度能力。
- 容灾技术及建设经验介绍

容灾演练流程

- 《依赖治理、灰度发布、故障演练，阿里电商故障演练系统的设计与实战经验》
    - 常见故障画像
    - 案例：预案有效性、预案有效性、故障复现、架构容灾测试、参数调优、参数调优、故障突袭、联合演练。

平滑启动

- 平滑重启应用思路 1.端流量（如vip层）、2. flush 数据(如果有)、3, 重启应用
- 《JVM安全退出（如何优雅的关闭java服务）》 推荐推出方式：System.exit，Kill SIGTERM；不推荐 kill-9；用 Runtime.addShutdownHook 注册钩子。
- 《常见Java应用如何优雅关闭》 Java、Srping、Dubbo 优雅关闭方式。

数据库扩展

读写分离模式

- 《Mysql主从方案的实现》
- 《搭建MySQL主从复制经典架构》
- 《Haproxy+多台MySQL从服务器(Slave) 实现负载均衡》
- 《DRBD+Heartbeat+Mysql高可用读写分离架构》

    - DRDB 进行磁盘复制，避免单点问题。

- 《MySQL Cluster 方式》

分片模式

- 《分库分表需要考虑的问题及方案》
    - 中间件： 轻量级：sharding-jdbc、TSharding；重量级：Atlas、MyCAT、Vitess等。
    - 问题：事务、Join、迁移、扩容、ID、分页等。
    - 事务补偿：对数据进行对帐检查;基于日志进行比对;定期同标准数据来源进行同步等。
    - 分库策略：数值范围；取模；日期等。
    - 分库数量：通常 MySQL 单库 5千万条、Oracle 单库一亿条需要分库。
- 《MySql分表和表分区详解》
    - 分区：是MySQL内部机制，对客户端透明，数据存储在不同文件中，表面上看是同一个表。
    - 分表：物理上创建不同的表、客户端需要管理分表路由。

服务治理

服务注册与发现

- 《永不失联！如何实现微服务架构中的服务发现？》
    - 客户端服务发现模式：客户端直接查询注册表，同时自己负责负载均衡。Eureka 采用这种方式。
    - 服务器端服务发现模式：客户端通过负载均衡查询服务实例。
- 《SpringCloud服务注册中心比较:Consul vs Zookeeper vs Etcd vs Eureka》
    - CAP支持：Consul（CA）、zookeeper（cp）、etcd（cp） 、euerka（ap）
    - 作者认为目前 Consul 对 Spring cloud 的支持比较好。
- 《基于Zookeeper的服务注册与发现》
    - 优点：API简单、Pinterest，Airbnb 在用、多语言、通过watcher机制来实现配置PUSH，能快速响应配置变化。

服务路由控制

- 《分布式服务框架学习笔记4 服务路由》
  - 原则：透明化路由
  - 负载均衡策略：随机、轮询、服务调用延迟、一致性哈希、粘滞连接
  - 本地路由有限策略：injvm(优先调用jvm内部的服务)，innative(优先使用相同物理机的服务),原则上找距离最近的服务。
  - 配置方式：统一注册表；本地配置；动态下发。

分布式一致

CAP 与 BASE 理论

- 《从分布式一致性谈到CAP理论、BASE理论》
  - 一致性分类：强一致(立即一致)；弱一致(可在单位时间内实现一致，比如秒级)；最终一致(弱一致的一种，一定时间内最终一致)
  - CAP：一致性、可用性、分区容错性(网络故障引起)
  - BASE：Basically Available（基本可用）、Soft state（软状态）和 Eventually consistent（最终一致性）
  - BASE理论的核心思想是：即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。

分布式锁

- 《分布式锁的几种实现方式》
  - 基于数据库的分布式锁：优点：操作简单、容易理解。缺点：存在单点问题、数据库性能够开销较大、不可重入；
  - 基于缓存的分布式锁：优点：非阻塞、性能好。缺点：操作不好容易造成锁无法释放的情况。
  - Zookeeper 分布式锁：通过有序临时节点实现锁机制，自己对应的节点需要最小，则被认为是获得了锁。优点：集群可以透明解决单点问题，避免锁不被释放问题，同时锁可以重入。缺点：性能不如缓存方式，吞吐量会随着zk集群规模变大而下降。
- 《基于Zookeeper的分布式锁》
  - 清楚的原理描述 + Java 代码示例。
- 《jedisLock—redis分布式锁实现》

    - 基于 setnx(set if ont exists)，有则返回false，否则返回true。并支持过期时间。

- 《Memcached 和 Redis 分布式锁方案》
    - 利用 memcached 的 add（有别于set）操作，当key存在时，返回 false。

分布式一致性算法

PAXOS

- 《分布式系列文章——Paxos算法原理与推导》
- 《Paxos-->Fast Paxos-->Zookeeper分析》
- 《【分布式】Zookeeper与Paxos》

Zab

- 《Zab：Zookeeper 中的分布式一致性协议介绍》

Raft

- 《Raft 为什么是更易理解的分布式一致性算法》
    - 三种角色：Leader（领袖）、Follower（群众）、Candidate（候选人）
    - 通过随机等待的方式发出投票，得票多的获胜。

Gossip

- 《Gossip算法》

两阶段提交、多阶段提交

- 《关于分布式事务、两阶段提交协议、三阶提交协议》

幂等

- 《分布式系统---幂等性设计》
    - 幂等特性的作用：该资源具备幂等性，请求方无需担心重复调用会产生错误。

- 常见保证幂等的手段：MVCC（类似于乐观锁）、去重表(唯一索引)、悲观锁、一次性token、序列号方式。

分布式一致方案

- 《分布式系统事务一致性解决方案》
- 《保证分布式系统数据一致性的6种方案》

分布式 Leader 节点选举

- 《利用zookeeper实现分布式leader节点选举》

TCC(Try/Confirm/Cancel) 柔性事务

- 《传统事务与柔性事务》
  - 基于BASE理论：基本可用、柔性状态、最终一致。
  - 解决方案：记录日志+补偿（正向补充或者回滚）、消息重试(要求程序要幂等)；“无锁设计”、采用乐观锁机制。

分布式文件系统

- 说说分布式文件存储系统-基本架构？
- 《各种分布式文件系统的比较》？
  - HDFS：大批量数据读写，用于高吞吐量的场景，不适合小文件。
  - FastDFS：轻量级、适合小文件。

唯一ID 生成

全局唯一ID

- 《高并发分布式系统中生成全局唯一Id汇总》
  - Twitter 方案（Snowflake 算法）：41位时间戳+10位机器标识（比如IP，服务器名称等）+12位序列号(本地计数器)
  - Flicker 方案：MySQL自增ID + "REPLACE INTO XXX:SELECT LAST_INSERT_ID();"
  - UUID：缺点，无序，字符串过长，占用空间，影响检索性能。
  - MongoDB 方案：利用 ObjectId。缺点：不能自增。
- 《TDDL 在分布式下的SEQUENCE原理》

- 在数据库中创建 sequence 表，用于记录，当前已被占用的id最大值。
- 每台客户端主机取一个id区间（比如 1000~2000）缓存在本地，并更新 sequence 表中的id最大值记录。
- 客户端主机之间取不同的id区间，用完再取，使用乐观锁机制控制并发。

一致性Hash算法

- 《一致性哈希算法》

设计思想 & 开发模式

DDD(Domain-driven Design - 领域驱动设计)

- 《浅谈我对DDD领域驱动设计的理解》
  - 概念：DDD 主要对传统软件开发流程(分析-设计-编码)中各阶段的割裂问题而提出，避免由于一开始分析不明或在软件开发过程中的信息流转不一致而造成软件无法交付（和需求方设想不一致）的问题。DDD 强调一切以领域（Domain）为中心，强调领域专家（Domain Expert）的作用，强调先定义好领域模型之后在进行开发，并且领域模型可以指导开发（所谓的驱动）。
  - 过程：理解领域、拆分领域、细化领域，模型的准确性取决于模型的理解深度。
  - 设计：DDD 中提出了建模工具，比如聚合、实体、值对象、工厂、仓储、领域服务、领域事件来帮助领域建模。
- 《领域驱动设计的基础知识总结》
  - 领域（Doamin）本质上就是问题域，比如一个电商系统，一个论坛系统等。
  - 界限上下文（Bounded Context）：阐述子域之间的关系，可以简单理解成一个子系统或组件模块。
  - 领域模型（Domain Model）：DDD的核心是建立（用通用描述语言、工具—领域通用语言）正确的领域模型；反应业务需求的本质，包括实体和过程；其贯穿软件分析、设计、开发 的整个过程；常用表达领域模型的方式：图、代码或文字；
  - 领域通用语言：领域专家、开发设计人员都能立即的语言或工具。
  - 经典分层架构：用户界面/展示层、应用层、领域层、基础设施层，是四层架构模式。
  - 使用的模式：
    - 关联尽量少，尽量单项，尽量降低整体复杂度。

        - 实体（Entity）：领域中的唯一标示，一个实体的属性尽量少，少则清晰。
        - 值对象（Value Object）：没有唯一标识，且属性值不可变，小二简单的对象，比如Date。
        - 领域服务（Domain Service）： 协调多个领域对象，只有方法没有状态(不存数据)；可以分为应用层服务，领域层服务、基础层服务。
        - 聚合及聚合根（Aggregate，Aggregate Root）：聚合定义了一组具有内聚关系的相关对象的集合；聚合根是对聚合引用的唯一元素；当修改一个聚合时，必须在事务级别；大部分领域模型中，有70%的聚合通常只有一个实体，30%只有2~3个实体；如果一个聚合只有一个实体，那么这个实体就是聚合根；如果有多个实体，那么我们可以思考聚合内哪个对象有独立存在的意义并且可以和外部直接进行交互；
        - 工厂（Factory）：类似于设计模式中的工厂模式。
        - 仓储（Repository）：持久化到DB，管理对象，且只对聚合设计仓储。

- 《领域驱动设计(DDD)实现之路》
    - 聚合：比如一辆汽车（Car）包含了引擎（Engine）、车轮（Wheel）和油箱（Tank）等组件，缺一不可。
- 《领域驱动设计系列（2）浅析VO、DTO、DO、PO的概念、区别和用处》

命令查询职责分离(CQRS)

CQRS — Command Query Responsibility Seperation

- 《领域驱动设计系列 (六)：CQRS》
    - 核心思想：读写分离（查询和更新在不同的方法中），不同的流程只是不同的设计方式，CQ代码分离，分布式环境中会有明显体现（有冗余数据的情况下），目的是为了高性能。
- 《DDD CQRS架构和传统架构的优缺点比较》
    - 最终一致的设计理念；依赖于高可用消息中间件。
- 《CQRS架构简介》
    - 一个实现 CQRS 的抽象案例。
- 《深度长文：我对CQRS/EventSourcing架构的思考》
    - CQRS 模式分析 + 12306 抢票案例

贫血，充血模型

- 《贫血，充血模型的解释以及一些经验》
  - 失血模型：老子和儿子分别定义，相互不知道，二者实体定义中完全没有业务逻辑，通过外部Service进行关联。
  - 贫血模型：老子知道儿子，儿子也知道老子；部分业务逻辑放到实体中；优点：各层单项依赖，结构清楚，易于维护；缺点：不符合OO思想，相比于充血模式，Service层较为厚重；
  - 充血模型：和贫血模型类似，区别在于如何划分业务逻辑。优点：Service层比较薄，只充当Facade的角色，不和DAO打交道、复合OO思想；缺点：非单项依赖，DO和DAO之间双向依赖、和Service层的逻辑划分容易造成混乱。
  - 肿胀模式：是一种极端情况，取消Service层、全部业务逻辑放在DO中；优点：符合OO思想、简化了分层；缺点：暴露信息过多、很多非DO逻辑也会强行并入DO。这种模式应该避免。
  - 作者主张使用贫血模式。

Actor 模式

TODO

响应式编程

Reactor

TODO

RxJava

TODO

Vert.x

TODO

DODAF2.0

- 《DODAF2.0方法论》

- 《DODAF2.0之能力视角如何落地》

Serverless

无需过多关系服务器的服务架构理念。

- 《什么是Serverless无服务器架构？》
    - Serverless 不代表出去服务器，而是去除对服务器运行状态的关心。
    - Serverless 代表一思维方式的转变，从“构建一套服务在一台服务器上，对对个事件进行响应转变为构建一个为服务器，来响应一个事件”。
    - Serverless 不代表某个具体的框架。
- 《如何理解Serverless？》
    - 依赖于 Baas （(Mobile) Backend as a Service） 和 Faas （Functions as a service）

Service Mesh

- 《什么是Service Mesh？》
- 《初识 Service Mesh》
- 《什么是Service Mesh？》

项目管理

架构评审

- 《架构设计之如何评审架构设计说明书》
- 《人人都是架构师：非功能性需求》

重构

- 《架构之重构的12条军规》

代码规范

TODO

代码 Review

制度还是制度！另外，每个公司需要根据自己的需求和目标制定自己的 check list

- 《为什么你做不好 Code Review？》
    - 代码 review 做的好，在于制度建设。
- 《从零开始Code Review》
- 《Code Review Checklist》
- 《Java Code Review Checklist》
- 《如何用 gitlab 做 code review》

RUP

- 《运用RUP 4+1视图方法进行软件架构设计》

看板管理

- 《说说看板在项目中的应用》

SCRUM

SCRUM - 争球

- 3个角色:Product Owner(PO) 产品负责人;Scrum Master（SM），推动Scrum执行;Team 开发团队。
- 3个工件：Product Backlog 产品TODOLIST，含优先级;Sprint Backlog 功能开发 TODO LIST；燃尽图；
- 五个价值观：专注、勇气、公开、承诺、尊重。
- 《敏捷项目管理流程-Scrum框架最全总结！》
- 《敏捷其实很简单3---敏捷方法之scrum》

敏捷开发

TODO

极限编程（XP）

XP - eXtreme Programming

- 《主流敏捷开发方法：极限编程XP》
    - 是一种指导开发人员的方法论。

- 4大价值：
  - 沟通：鼓励口头沟通，提高效率。
  - 简单：够用就好。
  - 反馈：及时反馈、通知相关人。
  - 勇气：提倡拥抱变化，敢于重构。
- 5个原则：快速反馈、简单性假设、逐步修改、提倡更改（小步快跑）、优质工作（保证质量的前提下保证小步快跑）。
- 5个工作：阶段性冲刺；冲刺计划会议；每日站立会议；冲刺后review；回顾会议。

结对编程

边写码，边review。能够增强代码质量、减少bug。

- 《结对编程》

PDCA 循环质量管理

P——PLAN 策划，D——DO 实施，C——CHECK 检查，A——ACT 改进

- 《PDCA》

FMEA管理模式

TODO

通用业务术语

TODO

技术趋势

TODO

政策、法规

TODO

法律

严格遵守刑法253法条

我国刑法第253条之一规定：

- 国家机关或者金融、电信、交通、教育、医疗等单位的工作人员，违反国家规定，将本单位在履行职责或者提供服务过程中获得的公民个人信息，出售或者非法提供给他人，情节严重的，处3年以下有期徒刑或者拘役，并处或者单处罚金。
- 窃取或者以其他方法非法获取上述信息，情节严重的，依照前款的规定处罚。
- 单位犯前两款罪的，对单位判处罚金，并对其直接负责的主管人员和其他直接责任人员，依照各该款的规定处罚。

最高人民法院、最高人民检察院关于执行《中华人民共和国刑法》确定罪名的补充规定（四）规定：触犯刑法第253条之一第1款之规定，构成“出售、非法提供公民个人信息罪”；触犯刑法第253条之一第2款之规定，构成“非法获取公民个人信息罪”

- 《非法获取公民个人信息罪》

架构师素质

- 《架构师画像》
  - 业务理解和抽象能力
  - NB的代码能力
  - 全面：1. 在面对业务问题上，架构师脑海里是否会浮现出多种技术方案；2. 在做系统设计时是否考虑到了足够多的方方面面；3. 在做系统设计时是否考虑到了足够多的方方面面；
  - 全局：是否考虑到了对上下游的系统的影响。
  - 权衡：权衡投入产出比；优先级和节奏控制；
- 《关于架构优化和设计，架构师必须知道的事情》
  - 要去考虑的细节：模块化、轻耦合、无共享架构；减少各个组件之前的依赖、注意服务之间依赖所有造成的链式失败及影响等。
  - 基础设施、配置、测试、开发、运维综合考虑。
  - 考虑人、团队、和组织的影响。
- 《如何才能真正的提高自己，成为一名出色的架构师？》

- 《架构师的必备素质和成长途径》
    - 素质：业务理解、技术广度、技术深度、丰富经验、沟通能力、动手能力、美学素养。
    - 成长路径：2年积累知识、4年积累技能和组内影响力、7年积累部门内影响力、7年以上积累跨部门影响力。
- 《架构设计师—你在哪层楼？》
    - 第一层的架构师看到的只是产品本身
    - 第二层的架构师不仅看到自己的产品，还看到了整体的方案
    - 第三层的架构师看到的是商业价值

团队管理

TODO

招聘

资讯

行业资讯

- 36kr
- Techweb

公众号列表

TODO

博客

团队博客

- 阿里中间件博客
- 美团点评技术团队博客

个人博客

- 阮一峰的网络日志
- 酷壳 - COOLSHELL-陈皓
- hellojava-阿里毕玄
- Cm's Blog
- 程序猿DD-翟永超-《Spring Cloud微服务实战》作者

综合门户、社区

国内：

- CSDN 老牌技术社区、不必解释。
- 51cto.com
- ITeye
    - 偏 Java 方向
- 博客园
- ChinaUnix
    - 偏 Linux 方向
- 开源中国社区
- 深度开源
- 伯乐在线
    - 涵盖 IT职场、Web前端、后端、移动端、数据库等方面内容，偏技术端。
- ITPUB
- 腾讯云— 云+社区
- 阿里云— 云栖社区
- IBM DeveloperWorks
- 开发者头条
- LinkedKeeper

国外：

- DZone
- Reddit

问答、讨论类社区

- segmentfault
  - 问答+专栏
- 知乎
- stackoverflow

行业数据分析

- 艾瑞网
- QUEST MOBILE
- 国家数据
- TalkingData

专项网站

- 测试:
  - 领测国际
  - 测试窝
  - TesterHome
- 运维:
  - 运维派
  - Abcdocker
- Java:
  - ImportNew
    - 专注于 Java 技术分享
  - HowToDoInJava
    - 英文博客
- 安全
  - 红黑联盟
  - FreeBuf
- 大数据
  - 中国大数据
- 其他专题网站:

    - DockerInfo
        - 专注于 Docker 应用及咨询、教程的网站。
    - Linux公社
        - Linux 主题社区

其他类

- 程序员技能图谱

推荐参考书

在线电子书

- 《深入理解Spring Cloud与微服务构建》
- 《阿里技术参考图册-研发篇》
- 《阿里技术参考图册-算法篇》
- 《2018美团点评技术年货（合辑）》70M
- InfoQ《架构师》月刊
- 《架构师之路》

纸质书

开发方面

- 《阿里巴巴Java开发手册》京东 淘宝

架构方面

- 《软件架构师的12项修炼：技术技能篇》京东 淘宝
- 《架构之美》京东 淘宝
- 《分布式服务架构》京东 淘宝
- 《聊聊架构》 京东 淘宝
- 《云原生应用架构实践》京东 淘宝
- 《亿级流量网站架构核心技术》京东 淘宝

- 《淘宝技术这十年》京东 淘宝
- 《企业IT架构转型之道-中台战略思想与架构实战》 京东 淘宝
- 《高可用架构（第1卷）》京东 淘宝

技术管理方面

- 《CTO说》京东 淘宝
- 《技术管理之巅》京东 淘宝
- 《网易一千零一夜：互联网产品项目管理实战》京东 淘宝

基础理论

- 《数学之美》京东 淘宝
- 《编程珠玑》京东 淘宝

工具方面

TODO

大数据方面

技术资源

开源资源

- github
- Apache 软件基金会

手册、文档、教程

国内：

- W3Cschool
- Runoob.com
    - HTML 、 CSS、XML、Java、Python、PHP、设计模式等入门手册。
- Love2.io

    - 很多很多中文在线电子书，是一个全新的开源技术文档分享平台。

- gitbook.cn
    - 付费电子书。
- ApacheCN
    - AI、大数据方面系列中文文档。

国外：

- Quick Code
    - 免费在线技术教程。
- gitbook.com
    - 有部分中文电子书。
- Cheatography
    - Cheat Sheets 大全，单页文档网站。
- Tutorialspoint
    - 知名教程网站，提供Java、Python、JS、SQL、大数据等高质量入门教程。

在线课堂

- 学徒无忧
- 极客时间
- segmentfault
- 斯达克学院
- 牛客网
- 极客学院
- 51CTO学院

会议、活动

- QCon
- ArchSummit
- GITC全球互联网技术大会

活动发布平台：

- 活动行

常用APP

- 极客时间

- 得到

找工作

- Boss直聘
- 拉勾网
- 猎聘
- 100Offer

工具

- 极客搜索
  - 技术文章搜索引擎。

代码托管

- Coding
- 码云

文件服务

- 七牛
- 又拍云

综合云服务商

- 阿里云
- 腾讯云
- 百度云
- 新浪云
- 金山云
- 亚马逊云(AWS)
- 谷歌云
- 微软云

VPS

- Linode
