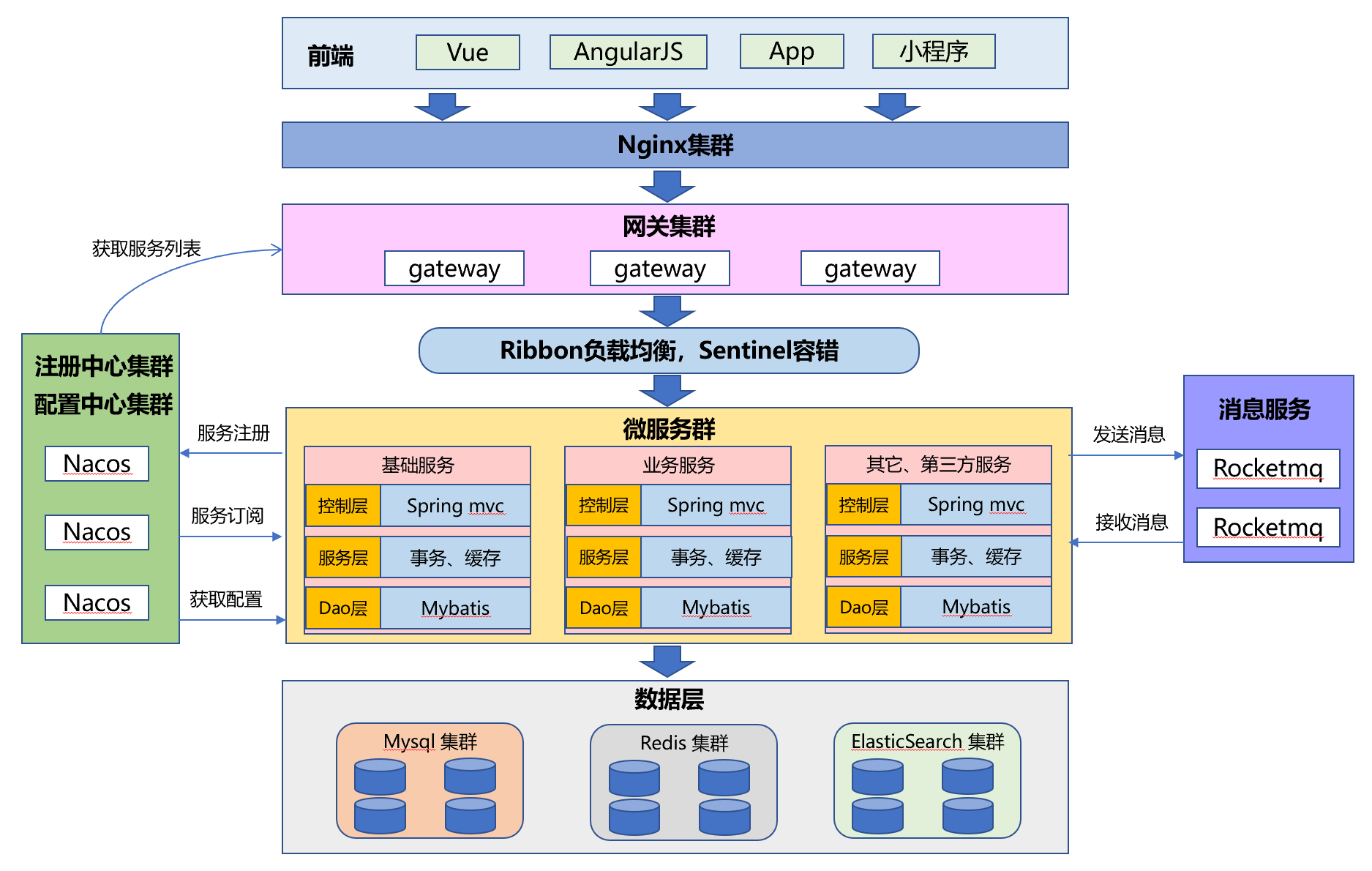
**Java技术学习规划整理**

# 技术架构图



# 并发编程

## 为什么要并发

并发是一种能并行运行多个程序或并行运行一个程序中多个部分的能力。如果程序中一个耗时的任务能以异步或并行的方式运行，那么整个程序的吞吐量和可交互性将大大改善。现代的PC都有多个CPU或一个CPU中有多个核。是否能合理运用多核的能力将成为一个大规模应用程序的关键。

## 并发的问题

线程有独自的调用栈，但是又能互相访问共享的数据。所以这里你会遇到两个问题，可见性和访问。

可见性问题发生于如果线程A先读取了某些共享数据，之后线程B对这些数据进行了修改，那么线程A可能看不到线程B对这数据的改动。

访问问题发生于于 多个线程同时访问修改同一个共享数据。

可见性及访问问题将导致：

活跃性失败：由于并发访问数据导致程序无任何反应。 譬如，死锁。

安全性失败：程序创建了错误的数据。

## 学习方向

### 内存模型

### 线程（池）

一:线程池：ThreadPoolExecutor、newFixedTreadPool、newCachedThreadPool、newSingleThreadPool、newScheduledThreadPool

二：Fork/join

三：CompletableFuture

四：纤程

### Lock 锁

一：辅助LockSupport，（AQS）AbstractQueuedSynchronizer

二：Lock，Condition，ReentrantLock、ReentrantReadWriteLock、StampedLock

### 原子类型

一：CAS 原理

二：AtomicBoolean、AtomicInteger、AtomicLong、AtomicReference、LongAdder、DoubleAdder、LongAccumulator、DoubleAccumulator

### 同步工具类

CountDownLatch、CyclicBarrier、Semphore

### 并发集合

ConcurrentHashMap、ConcurrentLinkedQueue、LinkedBlockingQueue、ArrayBlockingQueue、SynchronouseQueue

# Spring

## 为什么深入了解Spring

Spring 内部使用了非常多的好的设计思想，设计模式，值得我们去学习，可以运用到我们开发的项目中，这也是提高自己能力和价值的方法。去学习它的底层原理，假如只简单的使用框架，一旦遇到较为复杂的问题，一定是不知所措。

## 学习方向

### Spring Framework

IOC容器原理、BeanFactory

Bean生命周期、Bean 后置处理器 BeanPostProcessor、实始化 InitiallizingBean @PostConstruct， 销毁DisposableBean @PreDestroy SpringContext BeanFactoryPostProcessor、BeanDefinitionRegistryPostProcessor

### Spring MVC

MVC设计模式、DispatcherServlet、HandlerMapping、HandlerAdapter、ViewResolver、HandlerMethodArgumentResolver，@ControllerAdvice、@ExceptionHandler、HandlerInterceptor

### Spring Aop

AOP 实现原理、AOP 实际应用、Spring 事务控制

### SpringBoot

注解式配置、自动配置原理

### Spring5

新特性、响应式编程、WebFlux

# ORM（Mybatis）

## Mybatis优缺点

**优点**：

简单易学，容易上手（相比于Hibernate）基于SQL编程；

与各种数据库兼容；

可自定义插件（比如分页插件）；

能够与Spring很好的集成；

Mybatis相当灵活，SQL写在XML中，从程序代码中彻底分离，解除与程序代码的耦合，便于统一管理和优化，并可重用；提供XML标签，支持编写动态SQL语句，支持对象与数据库ORM字段映射，支持对象关联关系映射；

**缺点**：

SQL语句的编写工作量较大，尤其是字段多、关联表多时，更是如此，对开发人员编写SQL语句功底有一定要求；

SQL语句依赖数据库，导致数据库移植性差；

## 学习方向

### Spring-mybatis

### Mybatis-config.xml

### Mapper.xml

### Mapper.java

### 二级缓存

### Plugin

### Generator 反向工程

### MybatisPlus 第三方增强包

# 微服务

## 微服务优缺点

优点：

|  |  |  |
| --- | --- | --- |
|  | 微服务 | 单一架构 |
| 业务响应能力 | 实现未来平台可以更灵活快速地响应业务变化，可以实现新旧平台资产的整合和复用，降低整个平台拥有成本 | 业务需求的变更，都有可能牵动整个系统设计变更，整合和复用平台新旧资源复杂，非以用户的需求进化为核心，臃肿架构无法灵活快速响应业务变化 |
| 交付质量&效率 | 根据不同开发人员熟悉的业务环境，有针对性的划分工作任务，以便更好的分配人力、物力等资源，达到专业分工，提高交付质量，加快开发效率 | 对每个开发人员都要求熟悉整套架构，由于开发人员能力有高低，无法做到有针对性分配，影响交付质量和整体开发效率 |
| 开发难度 | 开发人员不需要对整个平台了解也可以对子系统单一业务模块开发，开发更易上手 | 开发人员必须了解整套系统架构才能进行开发，期间可能做大量重复工作，开发难度高，效率低下 |
| 未来扩展 | 松耦合关系、开放标准接口的采用，使其具有很好的维护性和可用性，当需求发生变化的时候，不需要修改提供业务服务的接口，只需要调整业务服务流程或者修改操作即可，非常适合未来多变的需求和系统系统扩展 | 紧耦合，无标准接口，各种不稳定性会加大运维成本高，10多年前的架构，不再适合当今需求个性化和多变的互联网发展 |
| 平台运维 | 每一个微服务都是独立部署的，可以进行快速迭代部署，根据各自服务需求选择合适的虚拟机和使用最匹配的服务资源要求的硬件 | 规模越大，启动时间越长，一个小功能的修改部署起来变得困难，必须重新部署整个应用 |

缺点：

1. 微服务应用作为分布式系统带来了复杂性。
2. 微服务增加了业务模块拆分难度。拆分粒度过细增加复杂度、服务数量多、服务间依赖关系复杂。拆分力度过粗使服务内部耦合度高，扩展性差
3. 系统拆分多个模块后增加关联查询难度。
4. 各服务之间数据一致性增加难度。

## 微服务框架Springcloud、Dubbo 比较

现阶段国内微服务框架使用最多的有Springcloud、Dubbo，甚至在企业中都有同时用到这两框架来构建微服务。

1. Dubbo 采用单一长连接和 NIO 异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。
2. Spring Cloud 使用 HTTP 协议的 REST API。Dubbo 支持各种通信协议，而且消费方和服务方使用长链接方式交互，通信速度上略胜 Spring Cloud，如果对于系统的响应时间有严格要求，长链接更合适。

3．Dubbo服务调用的方式是RPC，服务提供方与调用方接口依赖方式太强：我们需要将调用的抽象接口依赖到消费者项目中才能调用服务，这会导致在以后的开发、测试、版本管理上很麻烦。

4．SpringCloud调用的方式是REST，REST接口相比RPC更为轻量化，服务提供方和调用方的依赖只是依靠一纸契约，不存在代码级别的强依赖，当然REST接口也有缺点，很容易导致定义文档与实际实现不一致导致服务集成时的问题。

总结：

Dubbo 出生于阿里系，是阿里巴巴服务化治理的核心框架，已加入Apache顶级项目，并被广泛应用于中国各互联网公司；只需要通过 Spring 配置的方式即可完成服务化，对于应用无入侵，设计的目的还是服务于自身的业务为主。

Spring Cloud 自从发布到现在，仍然在不断的高速发展，几乎考虑了服务治理的方方面面，开发起来非常的便利和简单。

到底使用是 Dubbo 还是 Spring Cloud 并不重要，重点在于如何合理的利用微服务。不管是 Dubbo 还是 Spring Cloud 都是实现微服务有效的工具。

## Springcloud 学习方向

### 注册中心：Eureka（停止更新了），Nacos（Alibaba）

### 网关：Springcloud Getway

### 容断器：Hystrix（停止更新了），Sentinel（Alibaba）

### 负载均衡：Ribbon

### 服务调用：OpenFeign（推荐），RestTemplate

### 配置中心：Springcloud config，Apollo（携程）

### 服务链路跟踪：Sleuth（抽样采集），Zipkin（收集展示UI）

## Dubbo 学习方向

### Provider 服务提供方

### Consumer 服务消费方

### Registry 服务注册中心：Zookeeper，Nacos（推荐）

### Metadata 元素据中心（V2.7新版增加）：Consul，Etcd，Nacos，Redis，Zookeeper

### Monitor 服务蓝控中心

### Container 运行容器 （与Spring整合）

### Protocol 数据协议：Dubbo，Hessian，Json

# 数据库

## Redis 优缺点

### 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

### 支持丰富数据类型，支持string，list，set，sorted set，hash

### 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

### 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

## Redis学习方向

### Redis 底层原理

Redis 单线程为什么还能这么快？

因为它所有的数据都在内存中，所有的运算都是内存级别的运算（纳秒），而且单线程避免了多线程的切换（上下文切换）性能损耗问题。正因为 Redis 是单线程，所以要小心使用 Redis 指令，对于那些耗时的指令(比如keys)，一定要谨慎使用，一不小心就可能会导致 Redis 卡顿。

Redis 单线程如何处理那么多的并发客户端连接？

Redis的IO多路复用：redis利用epoll来实现IO多路复用，将连接信息和事件放到队列中，依次放到文件事件分派器，事件分派器将事件分发给事件处理器。

Nginx也是采用IO多路复用原理解决C10K问题。

### 安装配置

单机、主从、哨兵、集群模式，常用配置

### 命令使用

常用命令了解、在生产环境某些命令使用需特别谨慎

### 数据类型

String、List、Hash、Set、Sorted-Set

### 数据淘汰策略（解决热点数据）

为了限制最大使用内存，Redis 提供了配置参数 maxmemory 来限制内存超出期望大小。

当实际内存超出 maxmemory 时，Redis 提供了几种可选策略 (maxmemory-policy) 来让用户自己决定该如何腾出新的空间以继续提供读写服务。

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

### 持久化

持久化方式：RDB、AOF、混合持久化（V4.0新增）

### 应用场景

不同数据类型适用于不同场景

1. 字符串：缓存功能、计数器、共享session、分布式锁、限速（比如限制获取短信频率）
2. Hash：常用于\*\*用户信息\*\*等管理
3. List：用来储存多个有序的字符串，也可以充当栈和队列使用
4. Set：集合支持取交集、并集、差集，合理用好集合可以解决很多实际问题。
5. Sorted-Set：排行榜：有序集合经典使用场景。例如视频网站需要对用户上传的视频做排行榜，榜单维护可能是多方面：按照时间、按照播放量、按照获得的赞数等。

### Java客户端

Jedis、Spring-data-redis、lettuce、Redisson

# NIO

## Netty优势、应用场景

API 使用简单，开发门槛低；

功能强大，预置了多种编解码功能，支持多种主流协议；

定制能力强，可以通过 ChannelHandler 对通信框架进行灵活的扩展；

性能高，通过与其它业界主流的 NIO 框架对比，Netty 的综合性能最优；

成熟、稳定，Netty 修复了已经发现的所有 JDK NIO BUG，业务开发人员不需要再为 NIO 的 BUG 而烦恼；

社区活跃，版本迭代周期短，发现的BUG可以被及时修复，同时，更多的新功能会被加入；

经历了大规模的商业应用考验，质量已经得到验证。在互联网、大数据、网络游戏、企业应用、电信软件等众多行业得到成功商用，证明了它可以完全满足不同行业的商业应用。

## Netty学习方向

### 原理

线程模型：Reactor线程模型；

基于Buffer：可随意读取任意位置数据，可以零拷贝；

事件驱动模型：可扩展性好，分布式的异步架构，事件处理器之间高度解耦，可以方便扩展事件处理逻辑，高性能，基于队列暂存事件，能方便并行异步处理事件；

Netty 基于 Selector 对象实现 I/O 多路复用，通过 Selector 一个线程可以监听多个连接的 Channel 事件。

当向一个 Selector 中注册 Channel 后，Selector 内部的机制就可以自动不断地查询(Select) 这些注册的 Channel 是否有已就绪的 I/O 事件（例如可读，可写，网络连接完成等），这样程序就可以很简单地使用一个线程高效地管理多个 Channel。

### 核心组件

Bootstrap or ServerBootstrap

EventLoop

EventLoopGroup

ChannelPipeline

Channel

Future or ChannelFuture

ChannelHandler

### 解码器

MessageToMessageDecoder

DelimiterBasedFrameDecoder

FixedLengthFrameDecoder

LineBasedFrameDecoder

LengthFieldBasedFrameDecoder

### 编码器

MessageToByteEncoder

MessageToMessageEncoder

### 应用场景与案例

IM、Websocket、RPC、物联网、中间件、游戏服务端，应用容器

# Rocketmq

## Rocketmq优势

主流的MQ有很多，比如ActiveMQ、RabbitMQ、RocketMQ、Kafka、ZeroMQ等。

之前阿里巴巴也是使用ActiveMQ，随着业务发展，ActiveMQ IO 模块出现瓶颈，后来阿里巴巴 通过一系列优化但是还是不能很好的解决，之后阿里巴巴把注意力放到了主流消息中间件kafka上面，但是kafka并不能满足他们的要求，尤其是低延迟和高可靠性。

所以RocketMQ是站在巨人的肩膀上（kafka），又对其进行了优化让其更满足互联网公司的特点。它是纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。 RocketMQ目前在阿里集团被广泛应用于交易、充值、流计算、消息推送、日志流式处理、binglog分发等场景。

**数据可靠性**：RocketMQ支持异步实时刷盘，同步刷盘，同步复制，异步复制

**性能**：单机写入TPS单实例约7万条/秒，单机部署3个Broker，可以跑到最高12万条/秒，消息大小10个字节

Kafka的TPS跑到单机百万，主要是由于Producer端将多个小消息合并，批量发向Broker。 RocketMQ为什么没有这么做？

使用的Java语言，缓存过多消息，GC是个很严重的问题

Producer调用发送消息接口，消息未发送到Broker，向业务返回成功，此时Producer宕机，会导致消息丢失，业务出错

Producer通常为分布式系统，且每台机器都是多线程发送，我们认为线上的系统单个Producer每秒产生的数据量有限，不可能上万

缓存的功能完全可以由上层业务完成

**单机支持队列数**：Rocketmq单机支持最高5万个队列，单机可以创建更多话题，因为每个主题都是由一批队列组成，消费都集群规模和队列数成正比，队列越多，消费类集群可以越大

**消息投递实时性**：Rocketmq使用长轮询，同Push方式实时性一致，消息投递延时通常在几个毫秒

**消费失败重试**：Rocketmq消费失败支持定时重试，每次重试间隔时间顺延

**严格的消息顺序**：Rocketmq支持严格的消息顺序

**定时消息**：Rocketmq支持定时级别，定时级用户可定制

**分布式事务消息**：Rocketmq支持分布式事务消息

**消息查询**：Rocketmq支持根据消息标识查询消息，也支持根据滔滔息内容查询（发送消息时指定一个消息密钥，任意字符串，例如指定为订单号），消息查询对于定位消息丢失问题非常有帮助，例如某个订单处理失败，是消息没收到还是收到处理出错了。

**消息回溯**：Rocketmq支持按照时间来回溯消息，精度毫秒，例如从一天之前的某时某分某秒开始重新消费消息，总结：典型业备场景如consumer做订单分析，但是由于程序逻辑或者依赖的系统发生故障等原因，导致今天消费的消息全部无效，需要重新从昨天零点开始消费，那么以时间为起点的消息重放功能对于业务非常有帮助

**开发语言友好**：Rocketmq采用Java语言编写

## 应用场景

### 异步解耦

例如用户注册：用户提交注册账户信息，后续的注册短信与邮件等其它操作放入对应的消息队列Rocketmq中然后马上返回用户注册结果，由消息队列Rocketmq异步进行操作。

异步解耦是消息队列 RocketMQ 的主要特点，主要目的是减少请求响应时间和解耦。主要的使用场景就是将比较耗时而且不需要即时（同步）返回结果的操作作为消息放入消息队列。同时，由于使用了消息队列 RocketMQ，只要保证消息格式不变，消息的发送方和接收方并不需要彼此联系，也不需要受对方的影响，即解耦和。

### 削峰填谷

在秒杀或团队抢购活动中，由于用户请求量较大，导致流量暴增，秒杀的应用在处理如此大量的访问流量后，下游的通知系统无法承载海量的调用量，甚至会导致系统崩溃等问题而发生漏通知的情况。为解决这些问题，可在应用和下游通知系统之间加入消息队列 RocketMQ：

User<---->秒杀系统<----->Rocketmq<----->通知系统

处理流程：用户发起请求到秒杀系统处理----->系统按照满足秒杀条件的请求发送至消息队列Rocketmq------->下游通知系统订阅消息队列Rocketmq相关消息，再将秒杀成功的消息发送到用户

### 数据同步

比如多个服务模块都有保存关联的用户信息到各自模块，现在某一用户信息有更新，用户模块需要知通其它服务也要更新，这就可以使用到Rocketmq的广播消费模式，推送一条消息会被所有订阅的节点消费一次，相当于把用户信息同步到需要的每台机器上

## 学习方向

### 安装与配置

### 与Spring整合

### 组件

Producer

Consumer

Nameserver

Broker

### 消息订阅与发布

### 获取消息方式

Pull拉取，Push推送

### 消费方式

广播方式、集群方式

### 顺序消息

# EasticSerach

## ES优势

Elasticsearch（ES）是一个基于Lucene构建的开源、分布式、RESTful接口的全文搜索引擎。Elasticsearch还是一个分布式文档数据库，其中每个字段均可被索引，而且每个字段的数据均可被搜索，ES能够横向扩展至数以百计的服务器存储以及处理PB级的数据。可以在极短的时间内存储、搜索和分析大量的数据。通常作为具有复杂搜索场景情况下的核心发动机。

总结：易于部署；无需额外的软件即可扩展到数百个节点；内置Restful Api，上手快；开源+更新快+社区相当活跃。

## ES优秀案例

1．Github：采取ElasticSearch 来做PB级的搜索。 “GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”

2．维基百科：启动以elasticsearch为基础的核心搜索架构

3．SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”

4．百度：百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据。

5．淘宝等电商网站，新闻网站，OA办公系统等

## 相关名字说明

1. 节点(Node)和集群(Cluster)

集群是一个或多个节点（服务器）的集合， 这些节点共同保存整个数据，并在所有节点上提供联合索引和搜索功能。一个集群由一个唯一集群ID确定，并指定一个集群名（默认为“elasticsearch”）。该集群名非常重要，因为节点可以通过这个集群名加入集群，一个节点只能是集群的一部分。

2. Index(索引)

索引(index)类似于关系型数据库里的“数据库”——它是我们存储和索引关联数据的地方。索引名称必须是全部小写，不能以下划线开头，不能包含逗号。

3. Type(类型，在v6.0以后版本要移除Type)

在索引中，我们可以定义一个或多个类型。类型是索引的逻辑类别/分区，其语义完全由开发者决定。通常，为具有一组公共字段的文档定义类型。例如，假设开发者运行博客平台并将所有数据存储在一个索引中。在此索引中，我们可以为用户数据定义类型，为博客数据定义另一种类型，并为注释数据定义另一种类型。我们可以把索引理解成数据库文档中的表。

4. Document(文档)

文档是可索引信息的基本单元，以JSON表示。你可以用其来定义单个产品信息或是员工信息。我们可以把文档理解为数据库文档中的行列数据。在索引/类型中，您可以存储任意数量的文档。文档有几个共同不可缺的属性，分别为 \_index, \_type, \_id, 针对特定一个或一类文档进行操作时，必须指定这些属性。

5. Mapping(映射)

模式映射（schema mapping，或简称映射）用于定义索引结构。Elasticsearch在映射中存储有关字段的信息。映射在文件中以JSON对象传送。

6. Field(字段)

ElasticSearch里的最小单元 相当于数据的某一列，类似于json里一个键。

7. Shards(分片)

当有大量的文档时，由于内存的限制、硬盘能力、处理能力不足、无法足够快地响应客户端请求等，一个节点可能不够。在这种情况下，数据可以分为较小的称为分片（shard）的部分（其中每个分片都是一个独立的Apache Lucene索引）。每个分片可以放在不同的服务器上，因此，数据可以在集群的节点中传播。

当你查询的索引分布在多个分片上时，Elasticsearch会把查询发送给每个相关的分片，并将结果合并在一起。此外，多个分片可以加快索引。

8. Replica(副本)

为了提高查询吞吐量或实现高可用性，可以使用分片副本。副本（replica）只是一个分片的精确复制，每个分片可以有零个或多个副本。换句话说，Elasticsearch可以有许多相同的分片，其中之一被自动选择去更改索引操作。这种特殊的分片称为主分片（primary shard），其余称为副本分片（replica shard）。在主分片丢失时，例如该分片数据所在服务器不可用，集群将副本提升为新的主分片。

## 关系数据库和ES对应关系

|  |  |
| --- | --- |
| 关系型数据库 | Elasticsearch |
| 数据库Database | 索引Index，支持全文检索 |
| 表Table | 类型Type |
| 数据行Row | 文档Document，但不需要固定结构，不同文档可以具有不同字段集合 |
| 数据列Column | 字段Field |
| 模式Schema | 映射Mapping |

## 插件

中文分词插件：elasticsearch-analysis-ik

拼音分词插件：elasticsearch-analysis-pinyin

## 新建索引

## 检索文档

Match查询、term查询、match\_phrase查询、multi\_match查询、bool联合查询

## Java 客户端

Rest-high-level-client

## ELK

ElasticSearch+Logstash+Kibana 经典组合，用来快速搭建一个日志收集、分析、过滤、存储、搜索、可视化工具

# Git使用

## 安装git

## 配置git

## 创建一个新的本地仓库

## 检查状态

## 暂存

## 提交

## 远程仓库

## 分支

## 撤销与回退

## 合并与冲突解决

## 配置.gitignore

## 其它客户端使用

# Nginx 使用

# Maven使用

# Idea使用