什么是分布式系统？

要理解分布式系统，主要需要明白一下2个方面：

1.分布式系统一定是由多个节点组成的系统。

其中，节点指的是计算机服务器，而且这些节点一般不是孤立的，而是互通的。

2.这些连通的节点上部署了我们的节点，并且相互的操作会有协同。

分布式系统对于用户而言，他们面对的就是一个服务器，提供用户需要的服务而已，而实际上这些服务是通过背后的众多服务器组成的一个分布式系统，因此分布式系统看起来像是一个超级计算机一样。

例如淘宝，平时大家都会使用，它本身就是一个分布式系统，我们通过浏览器访问淘宝网站时，这个请求的背后就是一个庞大的分布式系统在为我们提供服务，整个系统中有的负责请求处理，有的负责存储，有的负责计算，最终他们相互协调把最后的结果返回并呈现给用户。

使用分布式系统主要有特点：

    1.增大系统容量。 我们的业务量越来越大，而要能应对越来越大的业务量，一台机器的性能已经无法满足了，我们需要多台机器才能应对大规模的应用场景。所以，我们需要垂直或是水平拆分业务系统，让其变成一个分布式的架构。

    2.加强系统可用。 我们的业务越来越关键，需要提高整个系统架构的可用性，这就意味着架构中不能存在单点故障。这样，整个系统不会因为一台机器出故障而导致整体不可用。所以，需要通过分布式架构来冗余系统以消除单点故障，从而提高系统的可用性。

    3.因为模块化，所以 系统模块重用度更高

    4.因为软件服务模块被拆分，开发和发布速度可以并行而变得更快

    5.系统扩展性更高

    6.团队协作流程也会得到改善

分布式系统的类型有三种：

    1.分布式处理，但只有一个总数据库，没有局部数据库

    2.分层式处理，每一层都有自己的数据库

    3.充分分散的分布式网络，没有中央控制部分，各节点之间的联系方式又可以有多种，如松散的联接，紧密的联接，动态的联接，广播通知式的联接等

什么是 JAVA 分布式应用？

一个大型的系统往往被分为几个子系统来做，一个子系统可以部署在一台机器的多个 JVM 上，也可以部署在多台机器上。但是每一个系统不是独立的，不是完全独立的。需要相互通信，共同实现业务功能。

一句话来说：分布式就是通过计算机网络将后端工作分布到多台主机上，多个主机一起协同完成工作。

实现分布式主要的方式

分布式应用用到的技术： 网络通信，基于消息方式的系统间通信和基于远程调用的系统间通信。

缺点： 就是会增加技术的复杂度。

基于消息的系统通信方式，主要是利用的网络协议，比如 TCP/IP 协议。

系统间的通信还需要对数据进行处理，比如同步 IO 和异步 IO。

远程调用实现系统间的通信:通过调用本地的java接口的方法来透明的调用远程java的实现。具体的细节有框架来实现。

基于Java自身技术实现消息方式的系统间通信：

基于Java自身包实现消息方式的系统间通信的方式有：

TCP/IP+BIO、TCP/IP+NIO、UDP/IP+BIO以及UDP/IP+NIO 4种方式

TCP/IP+BIO 在 Java 中可基于 Socket、ServerSocket 来实现 TCP/IP+BIO 的系统间通信。

Socket 主要用于实现建立连接及网络 IO 的操作，ServerSocket 主要用于实现服务器端端口的监听及 Socket 对象的获取。

多个客户端访问服务器端的情况下，会遇到两个问题:建立多个 socket 的，占用过多的本地资源，服务器端要承受巨大的来访量；创建过多的 socket，占用过多的资源，影响性能。

通常解决这种问题的办法是，使用 连接池，既能限制连接的数量，又能避免创建的过程，可以很大的提高性的问题。缺点就是竞争量大的时候造成激烈的竞争和等待。需要注意的是，要设置超时时间，如果不这样的话，会造成无限制的等待。

为了解决这个问题，采用一连接一线程的方式，同时也会带来副作用，内存占用过多。

TCP/IP 异步通信: JAVA NIO 通道技术实现。

JAVA 分布式知识体系介绍

附上某 JAVA 分布式学习目录，帮助了解分布式都有哪些东西

第 1 章 分布式 Java应用

1.1 基于消息方式实现系统间的通信

    1.1.1 基于 Java自身技术实现消息方式的系统间通信

    1.1.2 基于开源框架实现消息方式的系统间通信

1.2 基于远程调用方式实现系统间的通信

    1.2.1 基于 Java自身技术实现远程调用方式的系统间通信

    1.2.2 基于开源框架实现远程调用方式的系统间通信

第 2 章 大型分布式 Java应用与 SOA

2.1 基于 SCA实现 SOA平台

2.2 基于 ESB实现 SOA平台

2.3 基于 Tuscany实现 SOA平台

2.4 基于 Mule 实现 SOA平台

第 3 章 深入理解 JVM

3.1 Java代码的执行机制

    3.1.1 Java源码编译机制

    3.1.2 类加载机制

    3.1.3 类执行机制

3.2 JVM内存管理

    3.2.1 内存空间

    3.2.2 内存分配

    3.2.3 内存回收

    3.2.4 JVM 内存状况查看方法和分析工具

3.3 JVM线程资源同步及交互机制

    3.3.1 线程资源同步机制

    3.3.2 线程交互机制

    3.3.3 线程状态及分析

第 4 章 分布式应用与 SunJDK类库

4.1 集合包

    4.1.1 ArrayList

    4.1.2 LinkedList

    4.1.3 Vector

    4.1.4 Stack

    4.1.5 HashSet

    4.1.6 TreeSet

    4.1.7 HashMap

    4.1.8 TreeMap

    4.1.9 性能测试

    4.1.10 小结

4.2 并发包（ java.util.concurrent ）

    4.2.1 ConcurrentHashMap

    4.2.2 CopyOnWriteArrayList

    4.2.3 CopyOnWriteArraySet

    4.2.4 ArrayBlockingQueue

    4.2.5 AtomicInteger

    4.2.6 ThreadPoolExecutor

    4.2.7 Executors

    4.2.8 FutureTask

    4.2.9 Semaphore

    4.2.10 CountDownLatch

    4.2.11 CyclicBarrier

    4.2.12 ReentrantLock

    4.2.13 Condition

    4.2.14 ReentrantReadWriteLock

4.3 序列化 /反序列化

    4.3.1 序列化

    4.3.2 反序列化

第 5 章 性能调优

5.1 寻找性能瓶颈

    5.1.1 CPU消耗分析

    5.1.2 文件 IO 消耗分析

    5.1.3 网络 IO 消耗分析

    5.1.4 内存消耗分析

    5.1.5 程序执行慢原因分析

5.2 调优

    5.2.1 JVM 调优

    5.2.2 程序调优

    5.2.3 对于资源消耗不多，但程序执行慢的情况

第 6 章 构建高可用的系统

6.1 避免系统中出现单点

    6.1.1 负载均衡技术

    6.1.2 热备

6.2 提高应用自身的可用性

    6.2.1 尽可能地避免故障

    6.2.2 及时发现故障

    6.2.3 及时处理故障

    6.2.4 访问量及数据量不断上涨的应对策略

第 7 章 构建可伸缩的系统

7.1 垂直伸缩

    7.1.1 支撑高访问量

    7.1.2 支撑大数据量

    7.1.3 提升计算能力

7.2 水平伸缩

    7.2.1 支撑高访问量

    7.2.2 支撑大数据量

    7.2.3 提升计算能力