第一章 概述

1.1 分布式系统定义

一个分布式系统是由多个通过网络互联的独立自治的计算节点组成，这些计算节点为了完成共同目标基于消息传递机制进行相互协作。

要点：

- 多个计算节点：节点可以是进程、线程、抽象对象、组件、服务、单个计算机或一组计算机（图灵机）
- 网络互联：逻辑拓扑上一般认为是全连接网络
- 独立自治：独立 CPU、独立时钟、并发、独立错误
- 完成共同目标
- 相互协作
- 消息传递：消息传递模型，并非内存共享模型

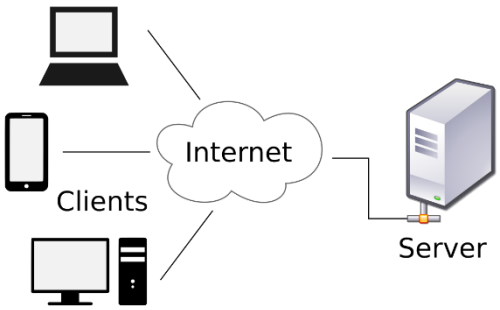
~~好处~~

1.2 分布式系统架构（5 种）

客户端-服务器(Client-Server)模式，主-从(Master-Slave)模式，总线模式，对等(Peer-to-Peer)模式，混合模式

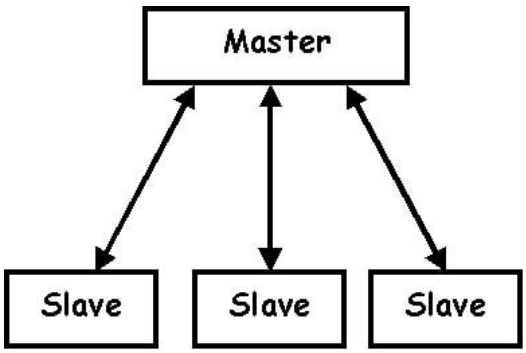
客户端-服务器(Client-Server)模式:

- 客户端发出服务请求，服务器端根据客户端请求参数完成实际运算，并将运算结果返回给客户端。
- 客户端运算任务清，服务器端运算任务重。
- 客户端生命周期短，服务器端生命周期长。
- 服务器端一般要应对并发问题。
- 客户端一般负责和用户进行交互。
- 瘦客户端/胖客户端(是否有大量的业务逻辑需要放在客户端)



主-从(Master-Slave)模式:

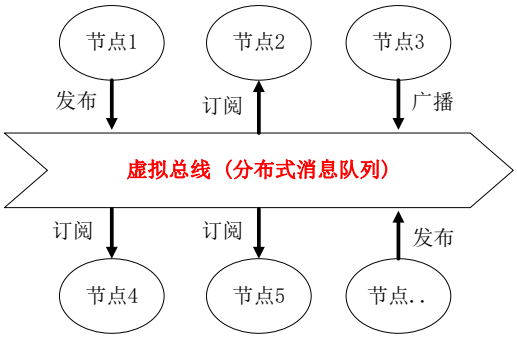
主节点（Master）负责将总计算任务分解为多个子任务分发给各个从节点（Slave，也叫 Worker 节点）完成主节点监视各个从节点的任务执行情况，将执行失败的任务调度给其它的从节点完成主节点在分配任务是会参考各个从节点的当前负载情况。



总线结构:

不同节点之间通过虚拟总线相连 消息发送者不必知道接收者是谁，接收者也不知道发送者是谁

发送者和接收者之间用异步方式通信 一种松耦合架构 不同节点完成不同功能，分工协作



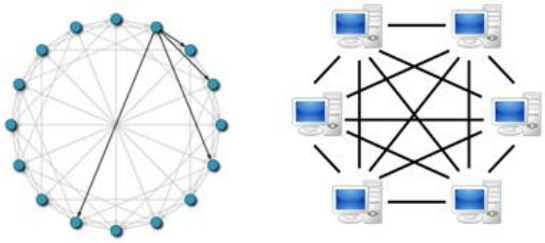
对等(Peer-to-Peer)模式:

系统中每个计算节点在任务分工上是完全对等的。

完全相同的软件在不同的计算机上运行，只是初始化参数不同

结构化 P2P：不同节点之间的交互模式遵循固定规律

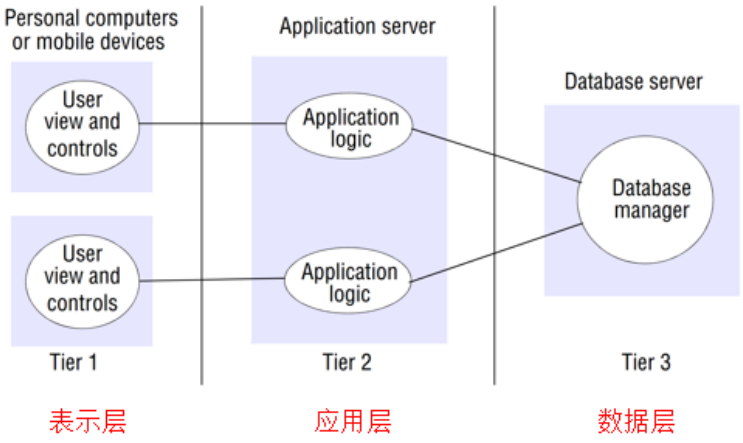
非结构化 P2P：不同节点之间的交互模式没有固定规律



1.3 服务器为集群时，划分为几层？为什么划分？各层功能是什么？好处有哪些？

随着企业信息的不断扩大，企业级应用不再满足于简单的两层系统，而是向着三层和多层体系结构发展。分层主要是为重用和便于管理，能够很好的适应需求的变化。

划分为三层：表示层 应用层 数据层



功能:

- 表现层：通俗讲就是展现给用户的界面，即用户在使用一个系统的时候他的所见所得。
- 应用层：又叫逻辑层，针对具体问题的操作，把复杂的业务关系细分为多项功能单一的服务，每项服务都执行一项特殊任务，这些服务可以用相对独立的服务组件来实现其功能。通过分布这些组件，可以平

衡数据处理负载，协调逻辑关系，调整业务规模和业务规则。

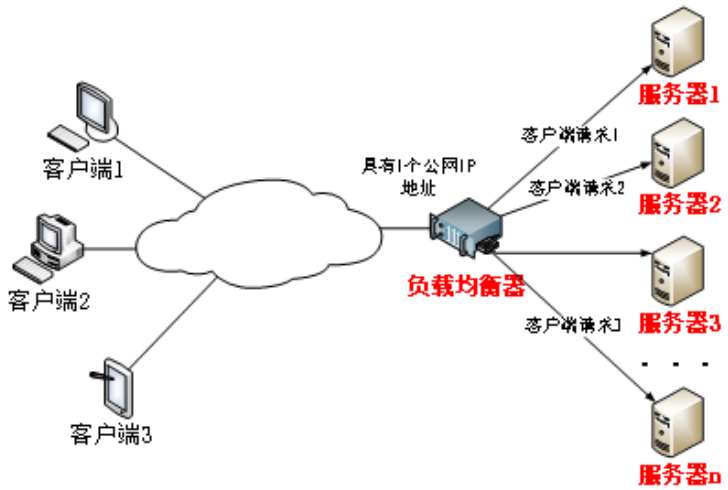
- 数据层：该层所做事务直接操作数据库，针对数据的增添、删除、修改、更新、查找等。

好处：

- **专注：开发人员可以只关注整个结构中的其中某一层；**
- 可以很容易的用新的实现来替换原有层次的实现；
- 可以降低层与层之间的依赖；
- 有利于标准化；
- 利于各层逻辑的复用。
- **扩展性强**。不同层负责不同的层面，接口定义好之后各个层不会相互影响。
- **安全性高。**用户端只能通过逻辑层来访问数据层，减少了入口点，把很多危险的系统功能都屏蔽了。
- 项目结构更清楚，分工更明确，有利于后期的维护和升级

1.4 服务器为集群时，使用了什么关键技术？实现该技术有哪些策略？

负载均衡技术：负载均衡服务器，就是用来把经过它的流量，按照某种方法，分配到集群中的各台服务器上。这样一来不仅可以承担更大的流量、降低服务的延迟，还可以避免单点故障造成服务不可用。



策略（7 种）：

- 随机：随机选择后端服务器
- 轮询：根据分布式系统配置文件中的顺序，依次把客户端的 Web 请求分发到不同的后端服务器。
- 固定权重值：把请求更多地分发到高配置的后端服务器上，把相对较少的请求分发到低配服务器。
- IP 哈希（基于一致性随机散列函数）：使用基于 IP 地址哈希的负载均衡方案，同一客户端连续的 Web 请求都会被分发到同一服务器进行处理。
- 最少 TCP 连接数：Web 请求会被转发到连接数最少的服务器上。
- 最小响应时间：连接到到那些通过响应最快之服务器
- 基于各服务器实际负载的动态负载均衡算法

1.5 IP 哈希（基于一致性随机散列函数）

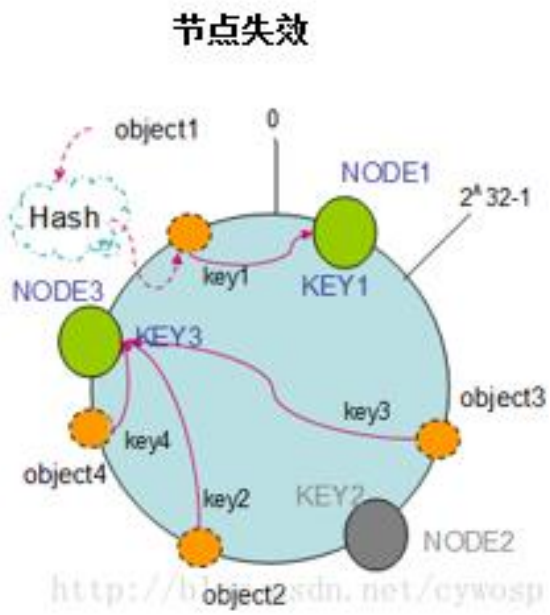
一致性哈希算法的目标是对于 K 个请求,节点的上下线只会引起 K/nodeTotal 的 key 重新映射,而在节点稳定的时候,同一个 key 的每次请求映射都是一样的

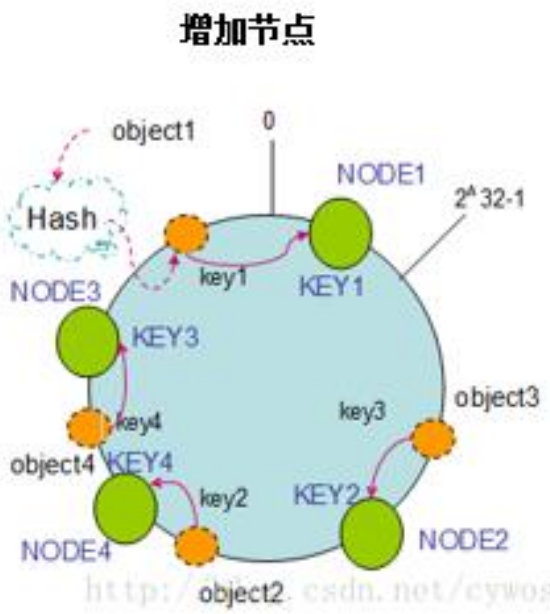
一致性哈希算法实现原理：

把数据通过一定的 hash 算法处理后映射到环上；将机器通过 hash 算法映射到环上，以顺时针的方向计算，

将所有对象存储到离自己最近的机器中。

（参考 https://blog.csdn.net/tingting256/article/details/52497749）



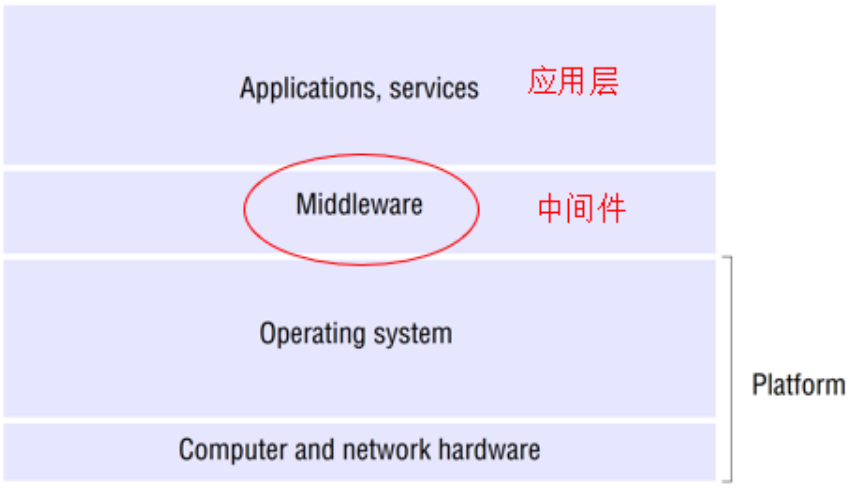


一致性哈希算法在保持了单调性的同时，还是数据的迁移达到了最小，这样的算法对分布式集群来说是非常合适的，避免了大量数据迁移，减小了服务器的的压力。

1.6 中间件（考点偏理解）

定义：

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。位于操作系统，计算机网络硬件等平台和应用层之间，管理计算机资源和网络通讯。是连接两个独立应用程序或独立系统的软件。相连接的系统，即使它们具有不同的接口，但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递。通过中间件，应用程序可以工作于多平台或 OS 环境。



常用中间件有哪些：

- 远程过程调用中间件
- 分布式对象中间件
- 分布式组件中间件

- 消息队列中间件
- Web 服务中间件
- P2P 中间件

中间件的作用:

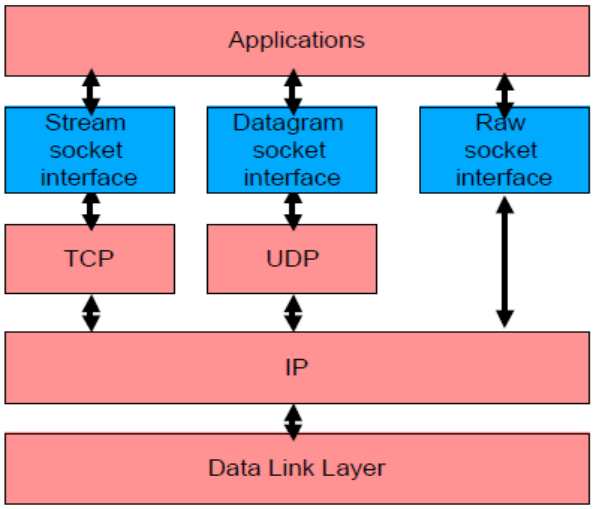
- 为开发者提供高层的编程抽象，屏蔽分布式系统底层的异构性和复杂性
- 提高交互操作性和可移植性
- 提供分布式系统的基础设施服务

第二讲 分布式节点之间的通信技术

2.1 Socket

定义:

传输层和网络层提供给应用层的标准化编程接口（或称为编程接口）



类型:

流式套接字(面向连接) 数据报套接字(面向无连接) 原始套接字

2.2 并发服务技术 适用场景

单机

- 基于多进程的并发服务技术（适合小数据量传送、弱相关的处理 ）
  优点：每个进程互相独立，不影响主程序的稳定性，子进程崩溃没关系；每个进程互相独立，不影响主程序的稳定性，子进程崩溃没关系；可以尽量减少线程加锁/解锁的影响，极大提高性能
  缺点：逻辑控制复杂，需要和主程序交互；多进程调度开销比较大；
- 基于多线程的并发服务技术（需要频繁创建销毁 需要进行大量计算 强相关的处理）
- 优点：无需跨进程边界；程序逻辑和控制方式简单；所有线程可以直接共享内存和变量等；线程方式消耗的总资源比进程方式好。
  缺点：线程之间的同步和加锁控制比较麻烦；一个线程的崩溃可能影响到整个程序的稳定性；进程切换和创建清楚耗时
- 基于线程池的并发服务技术：解决了线程创建销毁的问题 适用高并发、任务执行时间短的业务
- 事件驱动技术（多路复用技术）：解决了线程切换的问题 适合大量短事务

多机

- 负载均衡技术
  适用：负载均衡适用于高访问量的业务，提高应用程序的可用性和可靠性。

2.3 远程过程调用(Remote Method Invocation, RMI)

定义：

使应用程序可以像调用本地节点上的过程(子程序) 那样去调用一个远程节点上的子程序。

实现：

- RPC/RMI 中间件在调用者进程中植入 stub 模块，stub 模块作为远程过程的本地代理，并且暴露与远程过程相同的接口。
- RPC/RMI 中间件在被调用者进程中植入 skeleton 模块， skeleton 作为调用者在远程主机中的代理。
- stub 模块与 skeleton 模块利用 Socket 进行通信。
- skeleton 模块相当于 Client-Server 通信模式中的服务器端，要先于客户端运行，并且在某个 Socket 端口进行监听。

具体见 PPT 22 页-31 页 及其底下备注解释

四个核心组件：

分别是 Client，Client Stub，Server 以及 Server Stub，这个 Stub 可以理解为存根。

客户端(Client)，服务的调用方。

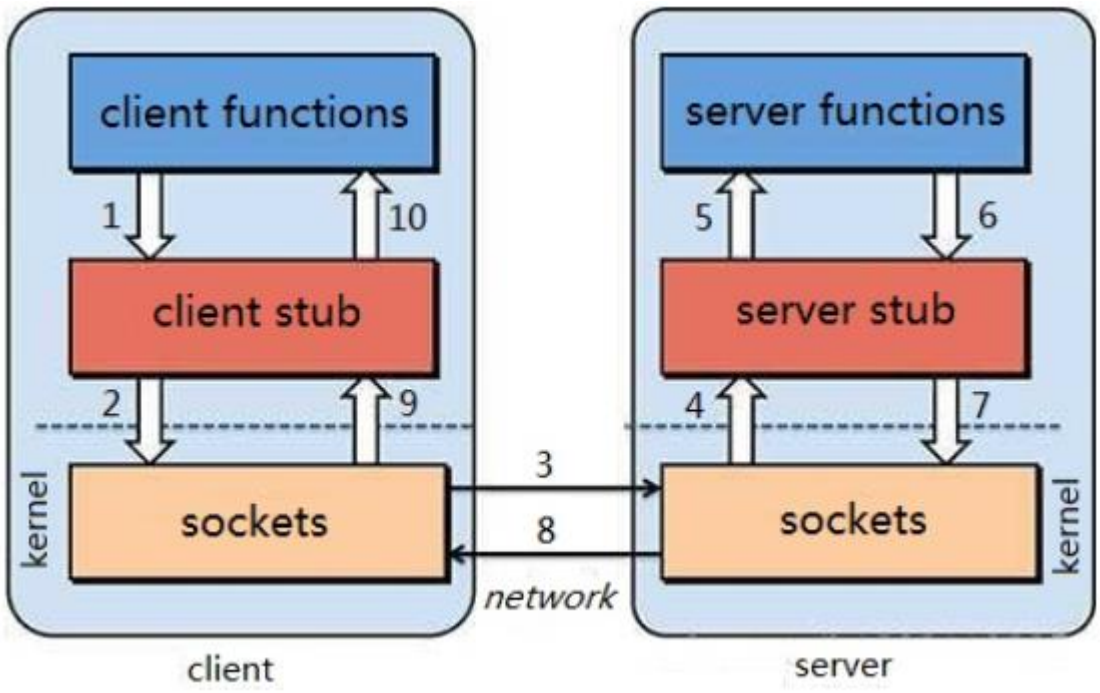
客户端存根(Client Stub proxy)，存放服务端的地址消息，再将客户端的请求参数打包成网络消息，然后通过 OS 网络远程发送给服务方。(TCP/UDP)

服务端(Server)，真正的服务提供者。

服务端存根(Server Stub skeleton)，接收客户端发送过来的消息，将消息解包，并分发 rpc 请求调用本地的方法。

RPC 调用过程：

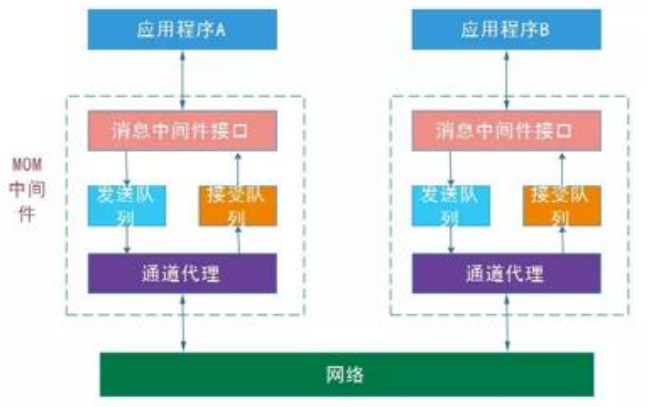
（参考 https://www.cnblogs.com/swordfall/p/8683905.html）



先注册
(1) 客户端（client）以本地调用方式（即以接口的方式）调用服务;
(2) 客户端存根（client stub）接收到调用后，负责将方法、参数等组装成能够进行网络传输的消息体（将消息体对象序列化为二进制）;
(3) 客户端通过 sockets 将消息发送到服务端;
(4) 服务端存根(server stub）收到消息后进行解码（将消息对象反序列化）;
(5) 服务端存根(server stub）根据解码结果调用本地的服务;
(6) 本地服务执行并将结果返回给服务端存根(server stub）;
(7) 服务端存根(server stub）将返回结果打包成消息（将结果消息对象序列化）;
(8) 服务端（server）通过 sockets 将消息发送到客户端;
(9) 客户端存根（client stub）接收到结果消息，并进行解码（将结果消息发序列化）;
(10) 客户端（client）得到最终结果。

2.4 面向消息中间件（MOM: Message Oriented Middleware）

定义:
提供了一种分布式消息队列服务，使得节点之间可以实现基于消息的形式灵活的异步通信。
异步的含义：发送方可以在任意时刻发出消息，不必等待接收方上线，更不必等待消息发送成功再做下一步工作；接收方不必以阻塞方式等待消息的到来。



基于 MOM 实现通信的优点:

- 异步通信，可以减少系统响应时间，提高吞吐量
- 分布式节点之间的解耦
- 保证消息的可靠递交，实现最终一致性
- 实现广播、组播和多对多通信
- 流量削峰和流控
- 支持 Push 模型和 Pull 模型

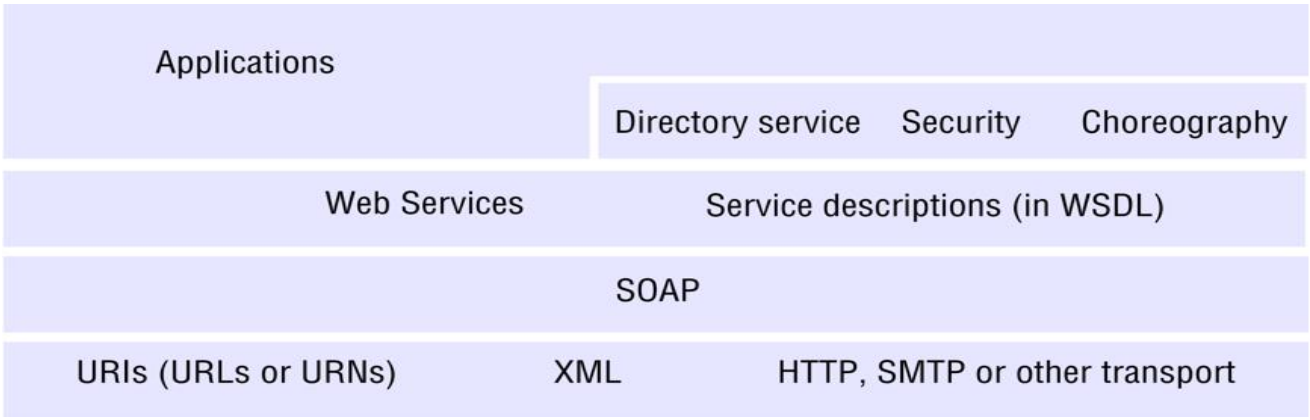
2.5 Web Service 技术（一种特殊的 RPC 技术）

定义:
为方便网络上不同节点之间互操作而定义的一套协议标准， 也可视为实现远程过程调用的一套协议标准。

Web Service 主要包含哪些标准协议?

- 消息编码标准(XML)
- 传输协议标准(HTTP、SMTP、TCP、UDP)
- 远程对象访问协议（即远程方法调用协议）：SOAP(Simple Object Access Protocol):
- Web 服务描述语言：WSDL(Web Services Description Language) （主要描述服务接口定义）

- 服务目录、服务注册、服务发现：UDDI(Universal Discovery Description and Integration)
- 安全相关标准：签名、加密、认证等
- 服务组合、服务编排



理解 WSDL SOA UDDI HTTP XML 等作用：

- 通过 XML 来实现消息描述，然后再通过 HTTP 实现消息传输。
- SOAP 是远程对象访问协议。SOAP 是一种简单的、轻量级的基于 XML 的机制，用于在网络应用程序之间进行结构化数据交换。SOAP 包括三部分:一个定义描述消息内容的框架的信封，一组表示应用程序定义的数据类型实例的编码规则，以及表示远程过程调用和响应的约定
- WSDL 表示 Web 服务描述语言。WSDL 文件是一个 XML 文档，用于说明一组 SOAP 消息以及如何交换这些消息,定义接口。
- UDDI(统一描述发现和集成) 提供一种发布和查找服务描述的方法。UDDI 数据实体提供对定义业务和服务信息的支持。WSDL 中定义的服务描述信息是 UDDI 注册中心信息的补充，定义注册中心。。

（参考 https://blog.csdn.net/zouyousu/article/details/83567545）

看看代码：

第三讲 分布式文件系统

3.1 分布式文件系统

定义：

分布式文件系统：将分布式系统中多个节点的存储资源整合在一起，向用户/应用程序呈现统一的存储空间和文件系统目录树。用户无需关心数据存储在哪个节点上。大文件被自动分块并分别存储到不同的节点上。

作用：

在用户看来就是一个大型的目录树。

安全机制：对读写文件有权限控制措施

可扩展性：支持通过增加节点扩充存储容量

高吞吐率：多副本可以提高数据吞吐率

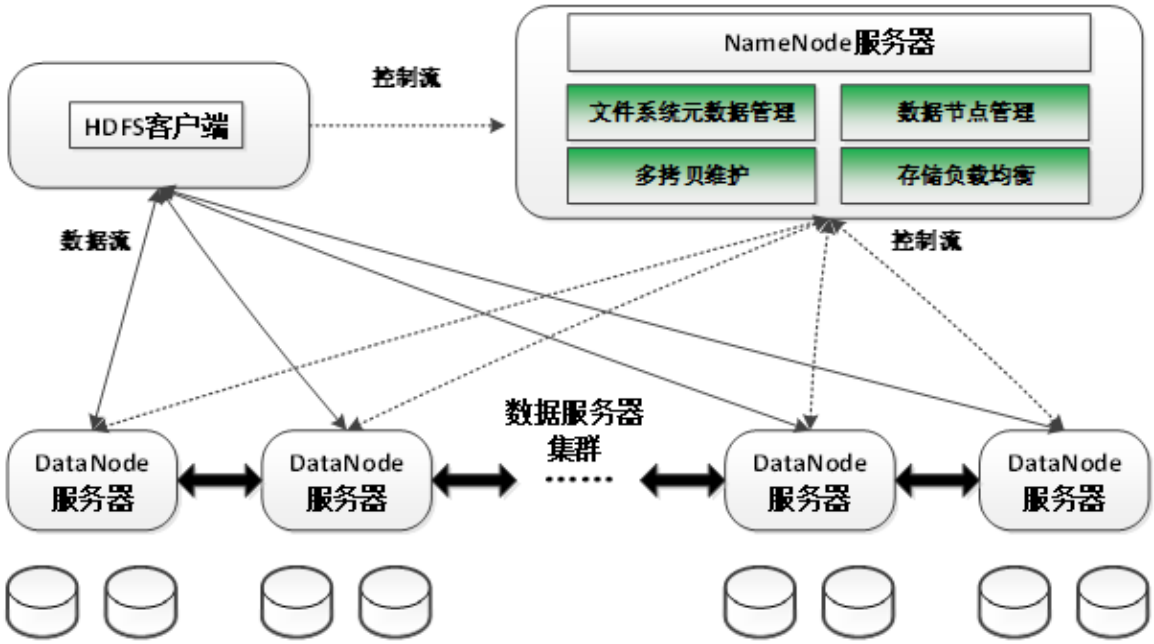
解决了大容量存储和负载均衡的问题。

工作原理：

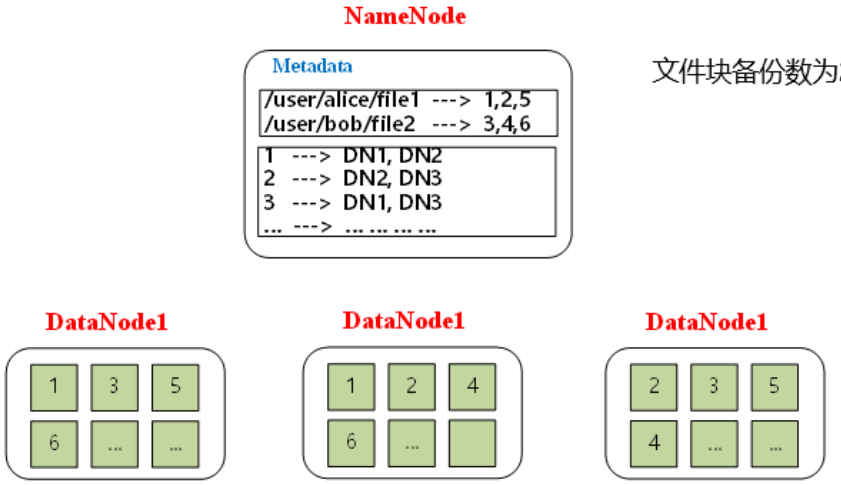
- 客户端向 NameNode 发出写文件请求。
- 检查是否已存在文件、检查权限。若通过检查，直接先将操作写入 EditLog，并返回输出流对象。

- client 端按 128MB 的块切分文件。
- client 将 NameNode 返回的分配的可写的 DataNode 列表和 Data 数据一同发送给最近的第一个 DataNode 节点，此后 client 端和 NameNode 分配的多个 DataNode 构成 pipeline 管道，client 端向输出流对象中写数据。client 每向第一个 DataNode 写入一个 packet，这个 packet 便会直接在 pipeline 里传给第二个、第三个…DataNode。
- 每个 DataNode 写完一个块后，会返回确认信息。
- 写完数据，关闭输输出流。
- 发送完成信号给 NameNode。

Namenode 节点流程及作用:
namenode 是整个文件系统的管理节点。它维护着 1.整个文件系统的文件目录树，2.文件/目录的元信息和每个文件对应的数据块列表。3.接收用户的操作请求。
namenode 包含两个文件: FsImage(元数据镜像文件。存储某一时段 NameNode 内存元数据信息)和 Editlog(操作日志文件)



NameNode 维护着 2 张表:
1 文件与数据块（block）列表的对应关系
2 数据块与被存储的结点的关系。



DataNode：负责存储 client 发来的数据块 block；执行数据块的读写操作；使用多备份策略。

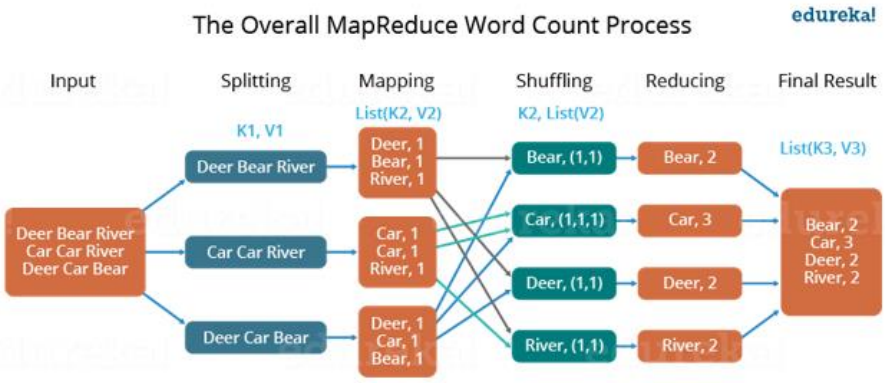
第四讲 MapReduce 模型和分布式计算框架

4.1 Mapreduce 设计算法即可，写伪代码，描述清楚

- Map 阶段：第一阶段并行，将输入文件划分成多个分区，每个分区都交给一个独立的 Map 子任务进行处理。强制要求每个 Map 子任务将输出规整为一系列<key, value>对的形式。
- 聚集混洗阶段：并不同 Map 子任务输出的<key, value>数组按照 key 进行聚集，聚集成数组 A = {<k1, [v1, v2, …]>, <k2, [v3, v4, …]>, <k3, [v5, v6, …]>, …}的形式。(按 key 排序)
- Reduce 阶段：第二阶段并行，将聚集之后的数组 A 划分成多个分区，每个分区都交给一个独立的 Reduce 子任务进行处理。 Reduce 子任务也将输出规整为一系列<key, value>对的形式。
- 如果 Reduce 阶段输出不是最终结果，还可以启动新一轮 MapReduce 过程。

4.2 样例设计：

- 单词计数统计



- 在网页数据库中统计不同网页被引用的频率。
  map 函数处理网页请求的日志文件，并输出<URL, 1>的键值对。reduce 函数把相同 URL 访问次数值相加到一起，输出<URL，总访问数>的键值对。

- 根据网页数据库生成关于每个网页的逆向链接统计。(<target, list(source)>)
  map 函数把 source 网页中每一个链接到 target URL 的结果输出为<target, source>。reduce 函数把所有和给定 target 相关的 source URLs 连接起来，并且输出<target, list(source)>。举个例子，搜索引擎中输入关键字，网络爬虫就会根据关键字(target)查找相关的网页并找到所有的链接，这些就是 list(source)。

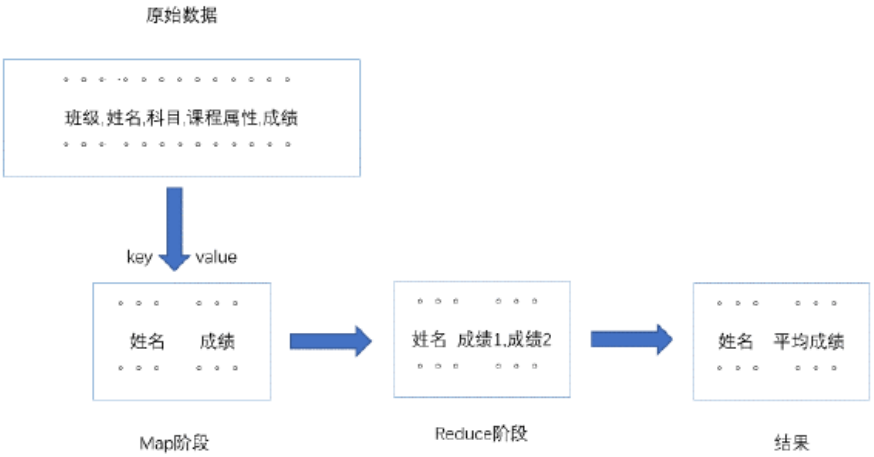
- 生成关于网页数据库的全文检索索引（倒排索引)。
  (1)这里存在两个问题：第一，<key,value>对只能有两个值，在不使用 Hadoop 自定义数据类型的情况下，需要根据情况将其中两个值合并成一个值，作为 key 或 value 值；第二，通过一个 Reduce 过程无法同时完成词频统计和生成文档列表，所以必须增加一个 Combine 过程完成词频统计。
  (2)这里讲单词和 URL 组成 key 值（如"MapReduce：file1.txt")，将词频作为 value，这样做的好处是可以利用 MapReduce 框架自带的 Map 端排序，将同一文档的相同单词的词频组成列表，传递给 Combine 过程，实现类似于 WordCount 的功能。
  (3)Combine 过程：经过 map 方法处理后，Combine 过程将 key 值相同的 value 值累加，得到一个单词在文档在文档中的词频，如果直接输出作为 Reduce 过程的输入，在 Shuffle 过程时将面临一个问题：所有具有相同单词的记录（由单词、URL 和词频组成）应该交由同一个 Reducer 处理，但当前的 key 值无法保证这一点，所以必须修改 key 值和 value 值。这次将单词作为 key 值，URL 和词频组成 value 值

（如"file1.txt：1"）。这样做的好处是可以利用 MapReduce 框架默认的 HashPartitioner 类完成 Shuffle 过程，将相同单词的所有记录发送给同一个 Reducer 进行处理。

- 海量数据的全文查找或关键字匹配

- 数据过滤
- 数据去重

- 数值数组计算总和、平均值、最大最小值、TopN
- 计算数字图像直方图
- 海量数据的随机采样
- 给出 child-parent 数据表，要求输出 grandchild-grandparent 数据集表



第五讲 时钟和分布式共识

5.1 抽象模型（理解）

交互模式

- 定义了在算法运行期间哪些节点之间有消息传递动作;
- 定义了通信模式：同步模式、异步模式

信道故障模式

- 消息是否会丢失
- 消息是否会乱序
- 消息传输延迟的上限

节点故障模式

- 失效停止模式
- 失效停止重启模式
- 拜占庭(Byzantine)模式

节点间的好友模式

- 全连接网络：每个节点可以向其他所有节点发送消息，也可以接收来自于其他任意节点的消息。
- 受限连接网络：每个节点只向协议规定的部分节点发送消息，只接收来自于部分节点的消息

5.2 同步交互与异步交互比较

同步交互模式

- 节点间的交互按周期进行，每个周期包含一次消息发送、和消息接收过程
- 隐含地对节点物理时钟的同步、消息传输最大延迟、消息最大处理时间做了假设。

异步交互模式

- 节点可以在任意时刻发送消息，也可以在任意时刻接收消息。
- 对消息传输延迟和消息处理时间没有任何假设

5.3 失效停止模式和拜占庭模式区别

失效停止模式(Fail-Stop Failure or Crash Failure)

- 节点一旦发生故障就停止工作，故障期间不发送任何消息
- 有的分布式协议也允许节点崩溃后可以重新启动，再次上线。此时节点崩溃之前内存中的状态全部消失，但持久化到非易失存储器中的状态还可以恢复。

拜占庭模式(Byzantine Failure)

- 节点一旦发生故障就有可能做出任意动作，例如发送非协议规定的错误消息，甚至故意扰乱分布式系统的消息。
- 网络安全系统设计多采用拜占庭模式，此时认为节点完全被黑客控制了。
- 一些安全系数要求极高的分布式系统设计中，也采用拜占庭模式。

5.4 物理时钟

实现方法：

方法一：每个节点都定期地与世界标准时间(UTC: Coordinated Universal Time)同步

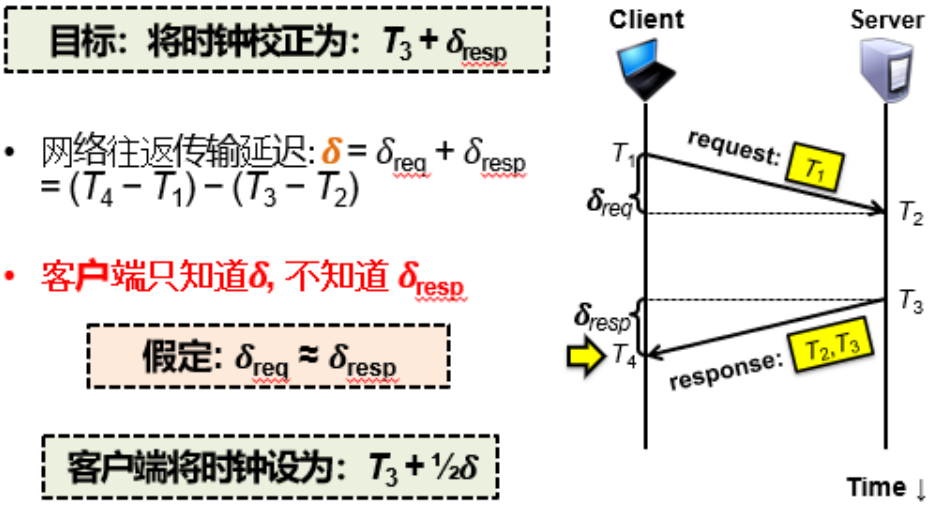
缺点：（1）成本高；（2）大部分计算机都放在室内，接收不到 UTC

方法二：每个节点定期地通过网络与时间服务器同步

缺点：时间服务器配备了 GPS 接收器。需要考虑传输延时

Cristian 时间同步算法（重要）

给出 T1-T4 计算或者推导公式：



5.5 逻辑时钟

思想：

- 分布式系统中很多问题的关键在于不同的节点对多个事件的发生顺序达成一致就可以，并且达成的顺序不能破坏因果关系。
- Lamport 提出的逻辑时钟概念就是为了解决分布式系统中事件的时序问题，并且能够保证如果事件 a 导致事件 b（记为：a➔b，表示 a 是 b 的潜在原因），则一定有 C(a)<C(b)。(C(a)为事件 a 发生的逻辑时间

戳， C(b)为事件 b 发生的逻辑时间戳）

- 系统内任意的两个消息 a、b 都可以基于时间戳<C(a), ID(a)> <C(b), ID(b)>比较先后，
  规则为：a 先于 b 当且仅当：C(a) < C(b) or C(a)=C(b) and ID(a)<ID(b)
- 有了消息比较先后的规则，分布式系统内所有消息都可以进行统一排序。
- 该排序的性质：保持了潜在因果关系：a 导致 b，则一定有 C(a)<C(b)
  但反过来不成立！因为可能 a,b 是不相关的。

节点维护逻辑时钟的算法
- 每个节点维护一个不断增加的整数作为本地逻辑时钟
- 节点内事件的发生（消息产生、消息发送、消息接收都是事件）会触发逻辑时钟的增长
- 节点 i 发出的每个消息都绑定一个二元组：
  <本地逻辑时钟 Ci，节点 ID 号 i>
- 每发生一个新事件就让本地逻辑时钟 Ci 加 1
- 假定节点 j 接收到一个消息 m，其时间戳为<Ci, i>，则节点 j 将自己的逻辑时间调整为：
  Cj 1 + max{ Cj， Ci }

5.6 分布式共识协议

介绍：
分布式系统由 n 个节点构成，其中最多有 f 个故障节点。每个节点 Pi 都有自己的提案 xi。这些节点之间通过运行“分布式共识协议”以达成一致的决策。

必须满足如下条件：
终止性：在有限时间内总能达成一致的决策。
共识性：不同节点最终决策的结果是相同的。
合法性：决策的结果必须等于某个进程提出的提案。

用分布式共识协议解决多数据库镜像一致性问题？
答：客户端向所有数据库服务器都发送 Update 消息，然后在服务器之间运行“分布式共识协议”就多个客户端 Update 消息的消息内容和先后顺序问题达成一致。

同步模型下的分布式共识协议

初始化：假定每个节点的提案都是自然数。每个节点 Pi 定义提案集合 Si={xi}
协议：每个节点按周期运行如下逻辑，共运行 f+1 个周期：
（1）将自己的集合 Si 中的尚未广播过的值广播给其他所有节点；
（2）从其他节点接收到的提案值都放入集合 Si（去掉重复的值）。
结束处理：
f+1 个周期后，每个节点都取集合 Si 中最大（或最小）的值作为最终提案。
定理：按照以上协议运行结束后，每个节点获得的集合 Si 是相同的。

异步模型下的分布式共识协议

FLP 定理：在异步通信模型下，只要有一个节点失效（失效停止模式，或拜占庭模式），就无法达成共识。

考试及文档说明:

- 8-9 道大题
- 以老师讲的 PPT 为准
- 我们是第一届，所以不要相信复印店所谓的分布式考试卷
- 问答题为主/考察你对于知识的理解，答题尽量使用专业术语表达

我们的实现一个 map 函数和 reduce 函数。我们看看 map 的方法:

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {…}

这里有三个参数，前面两个 Object key, Text value 就是输入的 key 和 value，第三个参数 Context context 这是可以记录输入的 key 和 value，例如：context.write(word, one);此外 context 还会记录 map 运算的状态。

对于 reduce 函数的方法:

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {…}

reduce 函数的输入也是一个 key/value 的形式，不过它的 value 是一个迭代器的形式 Iterable<IntWritable> values，也就是说 reduce 的输入是一个 key 对应一组的值的 value，reduce 也有 context 和 map 的 context 作用一致。

Main

Configuration conf = new Configuration(); 要初始化 Configuration，该类主要是读取 mapreduce 系统配置信息，这些信息包括 hdfs 还有 mapreduce

```
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
```

运行 WordCount 程序时候一定是两个参数，如果不是就会报错退出。

```
Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
```

第一行就是在构建一个 job，在 mapreduce 框架里一个 mapreduce 任务也叫 mapreduce 作业也叫做一个 mapreduce 的 job，而具体的 map 和 reduce 运算就是 task 了，这里我们构建一个 job，构建时候有两个参数，一个是 conf 这个就不累述了，一个是这个 job 的名称。
第二行就是装载程序员编写好的计算程序
第三行和第五行就是装载 map 函数和 reduce 函数实现类了

```
  job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

这个是定义输出的 key/value 的类型，也就是最终存储在 hdfs 上结果文件的 key/value 的类型。

```
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
```

第一行就是构建输入的数据文件，第二行是构建输出的数据文件，最后一行如果 job 运行成功了，我们的程序就会正常退出