分布式设计基础

提纲

- 什么是分布式设计
- 为什么要做分布式
- 分布式设计的优缺点
- 分布式设计系统基本分层
- 微服务概述
- 多租户设计概述
- 分布式设计简单案例
  - 分布式处理设计
  - 分布式存储设计
- 思考 - 标注平台哪些可以进行分布式重构？
- 框架推荐和学习

分布式系统概要

- 首先，本培训文档主要是从个人项目产品经验和理解出发来讲述的，大家若需要近一步的学习请多看看相关资料，文档中可能会出现与各个正规教程不一样的地方，以教程为主。
- 需要了解SOA、ESB、SAAS、微服务等相关知识
- 一般来说，分布式系统就是多台独立的计算机协同工作以完成任务,最明显的特征是组件分布在联网的计算机上，相同功能或不同功能组件之间通过传递消息通信和动作协调的系统。

为什么要做分布式

- 非分布式的设计，只能进行纵向扩展，如：增加主机的能力，但主机能力增强是有限的，价格呈指数性增长且能耗也随之增加，CPU的多内核多线程也无法完美使用。
- 分布式最基本能力是将纵向扩展变为横向扩展（伸缩性），因此天然具备以下能力：
  - 可以通过增加虚拟机、容器、物理机来进行扩展，增强并行能力来达到性能最优化，充分利用多核多线程的CPU特性。
  - 可以扩展也可以收缩，当业务对性能要求不高的情况下，通过手动或自动的方式，对物理机、虚拟机、容器进行收缩减少能耗。
  - 由于多个设备（虚拟机、容器、物理机）同时作用，特别是同功能组件，当出现单点故障时仅仅会影响那一条线的计算处理且后续派单不会再使用那条线直到恢复，因此提高了稳定性，属于高可用最优化设计。

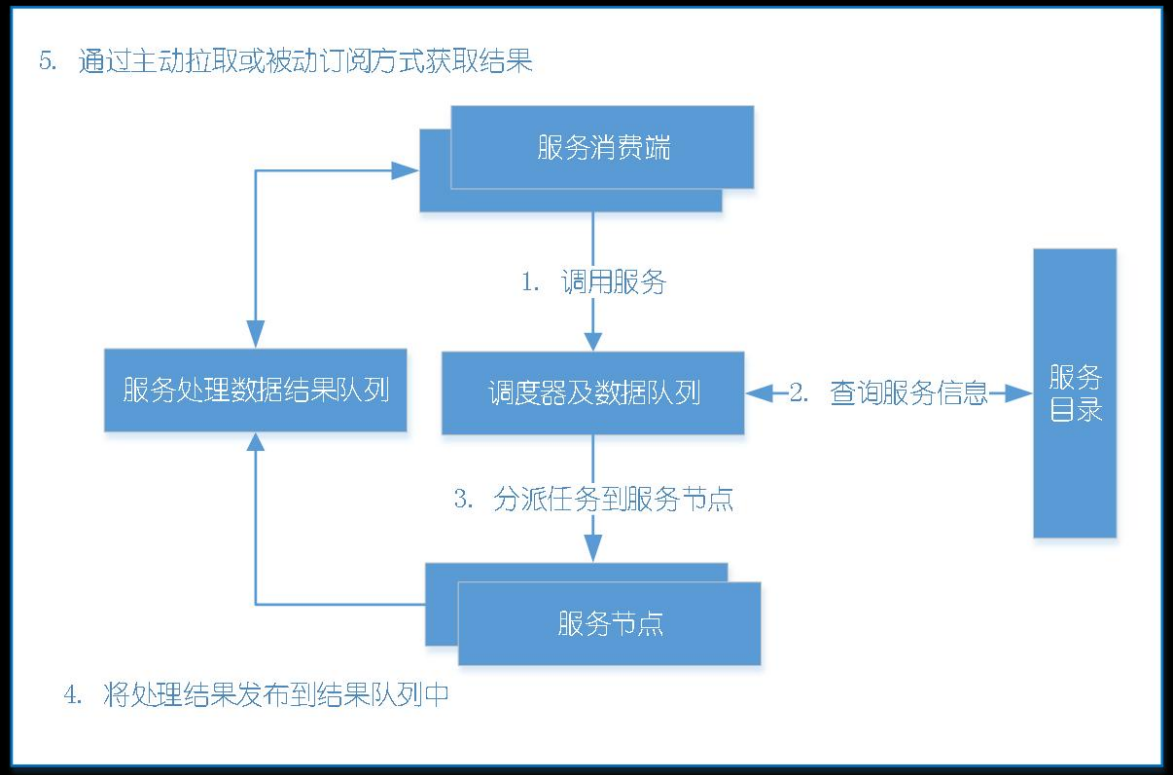
分布式优缺点

- 优点：
  - 高并发高性能
  - 高可用性
  - 可异构（容器化、物理、虚拟、不同CPU架构均可参与并行处理计算）
  - 支撑多种类客户端设计（多种类终端以服务方式调用设计出自己的操作风格，无需保证C/S架构一致）
- 缺点：
  - 设计有复杂度较高
  - 比起单点设计的系统更为庞大
  - 系统设计分层比传统系统设计层级更多（如：接口（interface）层、业务实现层会增加一个服务<->接口层(Service-interface)），复杂的则会增加多种类服务层，同时支持XAML、TCP/IP Socket、WSDL 等等
  - 分布式数据存储设计比单一数据存储设计更为复杂
  - 增加了调度器层和服务目录，调度器会根据服务节点的闲忙积分或权重进行任务分派、而服务目录则会为数据流和任务找到适合处理该数据的服务节点。
  - 开发量较大，特别是增加了服务层级后，需要在设计上尽量将细小功能拆分到服务上，原则上每个服务承担独立一个业务功能。
  - 大量采用异步处理、队列处理，若小业务量或小业务功能处理时及时性不如非分布式设计的系统。

分布式设计系统基本分层

- 服务目录及服务化
  - 每个细小功能都需要服务化且根据系统需求开发支持多种服务接口方式（一般来说目前为了节约开发成本，大量的企业都采用单一服务化原则即：WS或RESTful，但有些场景不太合适，比如单一大数据量的情况则会采用TCP/IP SOCKET，微软当时有个非常好的想法就是统一使用 .NET remoting service 标准，但没有企业响应且仅支持微软自己的.NET 产品系列，想法虽好但无法推广且缺少跨系统跨设备的基因，因此夭折了）
  - 每个服务都需要注册到服务目录库中（服务目录主表最少数据字段：服务名称、服务编号、服务地址编号、服务所在地址、服务接口类型、服务目录统计表最少数据字段：服务地址编号、调用次数、连通次数），当需要调用某个服务时，服务目录的服务将会告知调度器这些服务在哪些设备上，调度器会自动根据这些设备的闲忙状态和权重进行任务处理计算的分派（此处设备，包括：虚拟机、容器、物理机的任意一种）
  - 为了避免单点故障，服务目录之服务本身就可以通过NLB服务进行分布式部署
- 任务分派调度器
  - 最主要的工作是将需要处理的任务分派到各个服务节点上。
  - 为了避免单点故障，调度器本身就可以通过NLB服务进行分布式部署
- 服务节点
  - 每个服务都要放置到至少一个服务节点上面，根据调度器分派的任务进行处理。
- 服务处理结果队列
  - 用于存储服务处理结果
  - 当服务节点处理完后输出结果将会存储到这个队列中，消费端（服务调用者）如果订阅了相关消息则会推送给服务调用者，若没有订阅则需要服务调用者自行监控和拉取结果。

分布式设计系统基本分层



服务消费端：主要是调用服务和获取服务输出结果

调度器及数据队列：承担数据接收形成队列以及派单给服务节点进行任务执行。

服务目录：将注册到服务目录中的已发布的服务告知调度器可以将任务分派给哪些服务节点

服务节点：接收分派的任务并从指定队列中获取数据然后进行计算处理，再将结果发布到结果队列中（服务节点再设计中应该是发布者也是消费者）

服务处理数据结果队列：用于存储服务处理的结果等待拉取结果或者主动提交给消费者（订阅者）

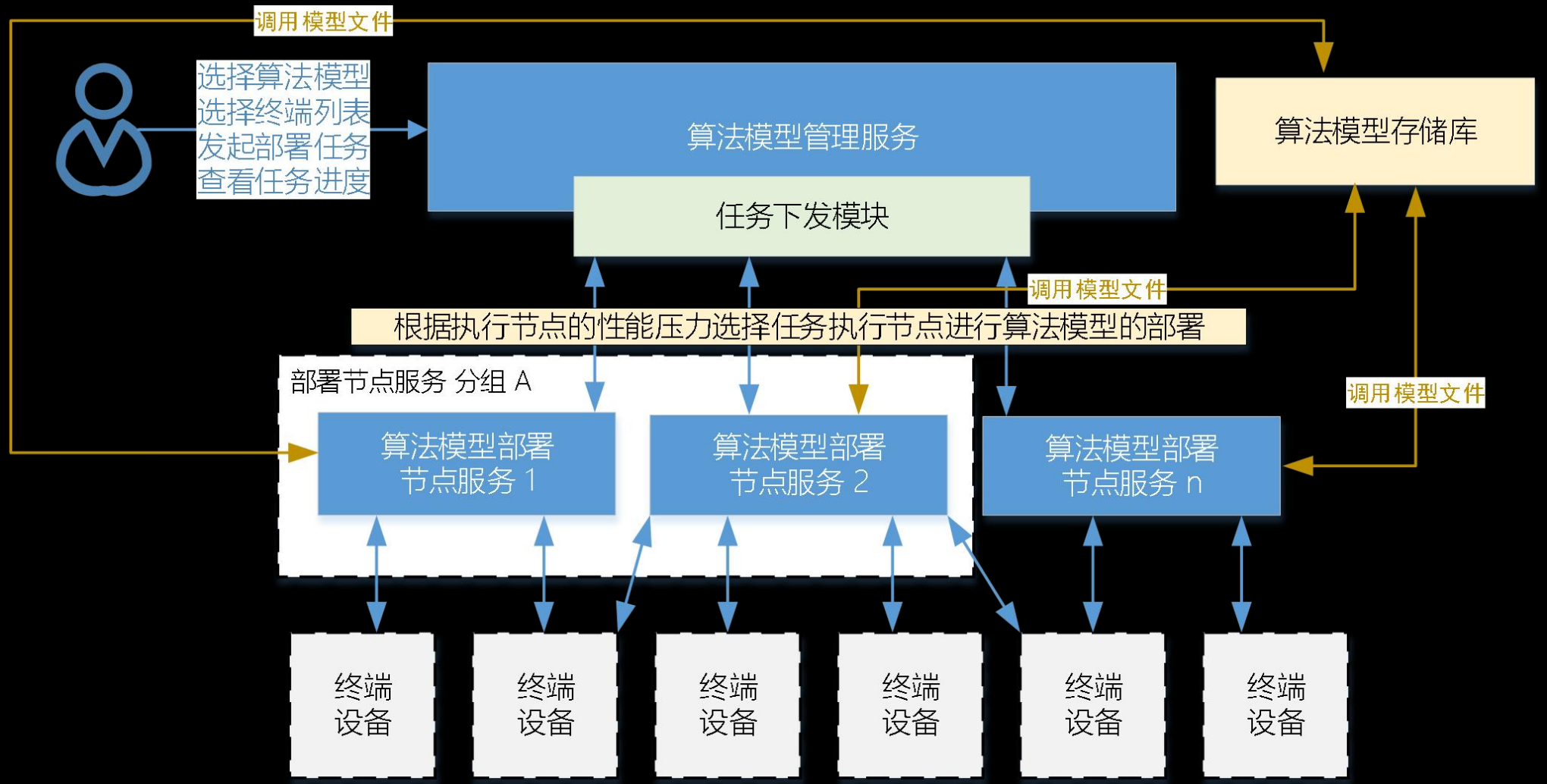
微服务概述

- 一种软件开发技术- 面向服务的体系结构（SOA）架构样式的一种变体，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够独立地部署到生产环境、类生产环境等。另外，应尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据上下文，选择合适的语言、工具对其进行构建。(来源于：wiki)
- 讨论
- 实际上是一种分布式计算（处理）的设计思路

多租户设计概述

- 在多租户技术中，租户（tenant）是指使用系统或电脑运算资源的客户，但在多租户技术中，租户包含在系统中可识别为指定用户的一切数据，包括帐户与统计信息（accounting data）、用户在系统中建置的各式数据、以及用户本身的定制化应用程序环境等，都属于租户的范围。租户使用供应商开发或建置的应用系统或运算资源，供应商所设计的应用系统会容纳数个以上的用户在同一个环境下使用，为了让多个用户环境能够在同一个应用程序与运算环境上使用，则应用程序与运算环境必须要特别设计，除了可以让系统平台允许多份相同的应用程序同时运行外，保护租户数据的隐私与安全也是多租户技术的关键之一。（来源于：wiki）
- 讨论
- 实际上是一种分布式存储的设计思路，需要保证系统程序资源共享但数据独立的一种设计模式。

分布式设计案例


调用 模型文件
选择算法模型
选择终端列表
发起部署任务
查看任务进度
算法模型管理服务
任务下发模块
算法模型存储库
调用 模型文件
根据执行节点的性能压力选择任务执行节点进行算法模型的部署
调用 模型文件
部署节点服务 分组 A
算法模型部署
节点服务 1
算法模型部署
节点服务 2
算法模型部署
节点服务 n
终端
设备
终端
设备
终端
设备
终端
设备
终端
设备
终端
设备

分布式存储概念

- 分布式存储主要是两个重要的概念：
  - 分布式存储一致性，多存储节点存储相同的内容，消费端调用存储时任意或者通过权重方式调取不同存储节点的数据，该设计中一定要保证数据一致性，也就是数据同步复制服务一定要跟上。（包含：文件存储系统、对象存储系统、数据库存储系统、内存数据库存储系统），这个主要基础架构上面来进行设计，目前有很多数据存储系统都支持分布式基础架构的部署，不是我们研发主要关心的内容。
  - 分布式存储策略，主要目的是减轻数据库处理压力，将数据通过策略分别存储到不同的存储单元，如：分库、分表等（主要指数据库存储系统 关系型数据库，如：SQL SERVER\MYSQL 等等），下一章节专门来讲分布式存储策略。
  - 以上两个概念并无冲突。

分布式存储设计概要

- 分布式存储设计，主要是针对数据库的分库设计、分表设计、表分区文件设计
  - 分库设计主要用于多租户进行数据隔绝，大多数SAAS平台都会进行数据库分库设计，分别是主库和租户库。
  - 在判断未来可能出现大数据量存储的情况下，我们需要考虑到分表分库，通常我们采用两种方案进行：
    - 使用编号顺序进行分库分表，通过编号计算HASH插入到分库分表并通过路由表（主要记录哪张表记录了哪些编号的数据，路由表一般放在主库中）指向该表再进行表操作。
    - 使用固定范围进行分库分表，同样需要通过查询路由指向该表再进行表操作。
  - 通过表分区文件的方式
    - 目前比较流行的关系型数据库应该都支持表分区，表分区是通过某个字段进行表分区配置函数对表的存储文件进行分区分文件，当查询时表分区配置文件负责联合操作，开发者无需考虑类似分表的路由查询，非常方便。

分布式存储分库分表示意图

思考 - 标注平台哪些可以进行分布式重构？

平台推荐和学习

- KAFKA
- Redis
- RocketMQ
- Zookeeper
- Nginx
- Hadoop

以上工具、框架和组件对应了我们前面所讲的内容，具体的资料可以从网络上查询