中间件期末复习

> 概念题(5道，20分)
>
> 简答题(5道，40分)
>
> 综合题(2道，20分)
>
> 设计题(1道，20分)

一、概念题

1. 什么是中间件?

中间件(Middleware)是一类提供系统软件和应用软件之间连接、便于软件各部件之间的沟通的软件，应用软件可以借助中间件在不同的技术架构之间共享信息与资源。中间件位于客户机服务器的操作系统之上，管理着计算资源和网络通信。

2. 什么是分布式互操作?

在分布式软件系统中，有许多独立的、网络连接的、通讯的，并且物理上分离的计算节点，有着通用的数据结构和传输标注设置，使之可以互换数据和执行命令的解决方案。软件的互操作往往通过Java/.NET/Corba/COM等标准和通用接口，或者通过专门的适配系统实现两种异构系统之间的互操作。

3. 什么是对象请求总线?

对象请求总线(ORB,Object request broker)，又称对象请求代理，是一种中间件，它允许通过计算机网络从一台计算机到另一台计算机进行程序调用，通过远程过程调用提供位置透明性。ORB 促进了分布式对象系统的互操作性，使此类系统能够通过将来自不同供应商的对象拼凑在一起来构建，而不同的部分通过 ORB 相互通信。

4. 什么是负载均衡?

负载平衡(load balancing)是一种电子计算机技术，用来在多个计算机（计算机集群）、网络连接、CPU、磁盘驱动器或其他资源中分配负载，以达到优化资源使用、最大化吞吐率、最小化响应时间、同时避免过载的目的。

5. 什么是容器？容器和VM的关系？为什么需要容器?

容器是应用服务器中位于组件和平台之间的接口集合。

虚拟机(VM)是计算机系统的仿真，它可以在计算机硬件上运行看似很多单独的计算机，每个VM都需要自己的底层操作系统，并虚拟化硬件；**容器**不像VM那样虚拟化底层计算机，而只是虚拟化操作系统，其位于物理服务器及其主机系统之上(通常是Linux或Windows)，每个容器共享操作系统内核。

使用容器可以减少IT管理资源，缩小快照大小，更快地启动应用程序，减少和简化安全更新，减少传输、迁移、上传工作负载的代码。

6. 什么是微服务架构？什么是SOA架构?

微服务(Microservices)是一种软件架构风格，它是以专注于单一责任与功能的小型功能区块为基础，利用模块化的方式组合出复杂的大型应用程序，各功能区块使用与语言无关的API集相互通信。

面向服务的体系结构(SOA)是一种分布式运算的软件设计方法。调用者(软件)可以透过网络上的通用协议调用另一个应用软件组件执行、运作，让调用者获得服务，因此可以跨越厂商、产品和技术。

7. 什么是组件？什么是面向组件编程?

组件(Component)是对数据和方法的简单封装。

面向组件编程(CBD)是针对系统的广泛功能，进行关注点分离的软件工程方式。此方式是以复用为基础的作法，定义、实现许多松耦合的独立组件(Component)，再将组件组合成为系统。

8. 几个ASS的概念

LaaS(基础设施即服务)是提供消费者处理、储存、网络以及各种基础运算资源，以部署与执行操作系统或应用程序等各种软件。IaaS 是云服务的最底层，主要提供一些基础资源。

PaaS(平台即服务)是一种云计算服务，提供用户将云端基础设施部署与创建至客户端，或者借此获得使用编程语言、程序库与服务。用户不需要管理与控制云端基础设施，但需要控制上层的应用程序部署与应用托管的环境。

SaaS(软件即服务)是一种软件交付模式[3]。在这种交付模式中，软件仅需通过网络，不须经过传统的安装步骤即可使用，软件及其相关的数据集中托管于云端服务。用户通常使用精简客户端，一般即经由网页浏览器来访问、访问软件即服务。

CaaS(通讯即服务)是将传统电信的能力如消息、语音、视频、会议、通信协同等封装成API或者SDK通过互联网对外开放，提供给第三方使用，将电信能力真正作为服务对外提供。

9. 一些英文缩写

RPC(Remote Procedure Call)：**远程过程调用**，是分布式计算中的一个计算机通信协议，该协议允许运行于一台计算机的程序调用另一个地址空间（通常为一个开放网络的一台计算机）的子程序，而程序员就像调用本地程序一样，无需关注细节。

DDL(Distributed Description Logic)：**分布式描述逻辑**，是描述逻辑的一种特例。在DDL中，整个逻辑系统由一组DL单元组成，相互之间用Bridge Rule相互连接。

IDL(Interface Definition Language)：**是用于描述分布式对象接口的定义语言**，利用IDL进行接口定义之后，就确定了客户端与服务器之间的接口，这样即使客户端和 服务器 独立进行开发，也能够正确地定义和调用所需要的分布式方法。

GIOP(General Inter-ORB Protocol)：**通用对象请求代理间通信协议**，是分布式计算领域的一种抽象协议，对象请求代理（ORB）通过该协议进行通信。

10. 两段锁协议和两阶段提交

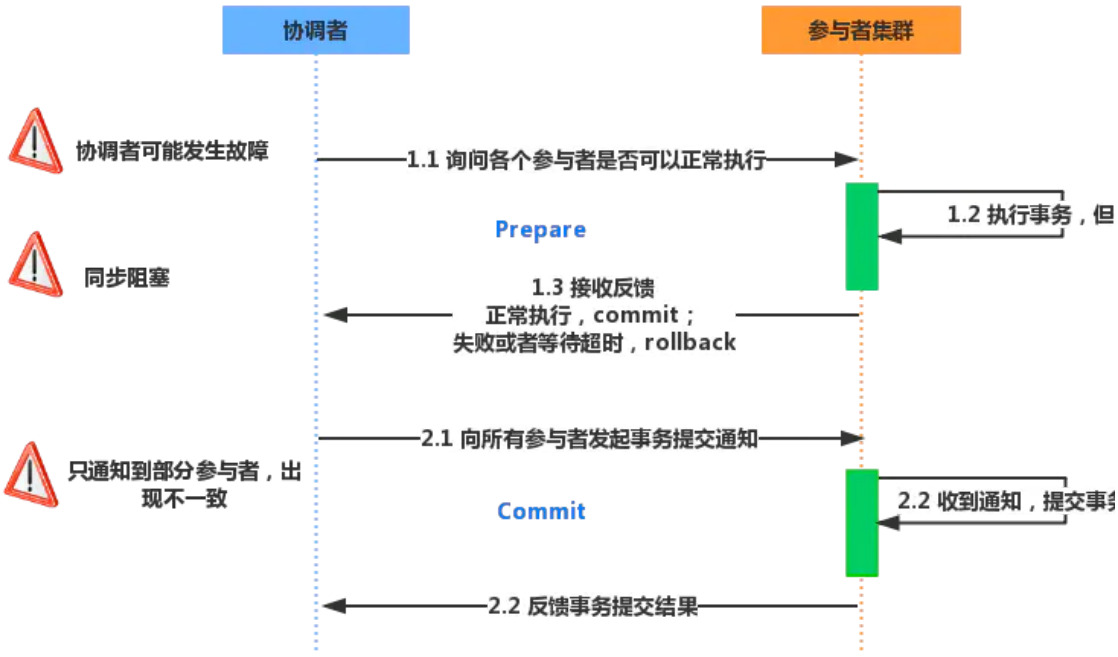
两端锁协议：所有的事务必须分两个阶段对数据项加锁和解锁。即**事务分两个阶段**，第一个阶段是获得锁。事务可以获得任何数据项上的任何类型的锁，但是不能释放；第二阶段是释放锁，事务可以释放任何数据项上的任何类型的锁，但不能申请。

获得锁的阶段称为扩展阶段，在这个阶段可以进行加锁操作，**对任何数据进行读操作之前要申请获得S锁，在进行写操作之前要申请并获得X锁，加锁不成功，则事务进入等待状态**，直到加锁成功才继续执行。加锁后无法解锁。

释放锁的阶段称为收缩阶段，当事务释放一个锁后，事务进入收缩阶段，在该阶段只能解锁不能加锁。

在分布式系统中，为了让每个节点都能够感知到其他节点的事务执行状况，需要引入一个中心节点来统一处理所有节点的执行逻辑，这个中心节点叫做**协调者**，被中心节点调度的其他业务节点叫做**参与者**

两阶段提交(2PC)：2PC将分布式事务分成了两个阶段，两个阶段分别为提交请求(投票)和提交(执行)。协调者根据参与者的响应来决定是否需要真正地执行事务。



提交请求(投票)阶段

- 协调者向所有参与者发送prepare请求与事务内容，询问是否可以准备事务提交，并等待参与者的响应。
- 参与者执行事务中包含的操作，并记录undo日志(用于回滚)和redo日志(用于重放)，但不真正提交。
- 参与者向协调者返回事务操作的执行结果，执行成功返回yes，否则返回no。

提交(执行)阶段

- 成功情况：若所有参与者都返回yes，说明事务可以提交
  - 协调者向所有参与者发送commit请求
  - 参与者收到commit请求后，将事务真正地提交上去，并释放占用的事务资源，并向协调者返回ack
  - 协调者收到所有参与者的ack消息，事务成功完成
- 失败情况：若有参与者返回no或者超时未返回，说明事务中断，需要回滚
  - 协调者向所有参与者发送rollback请求
  - 参与者收到rollback请求后，根据undo日志回滚到事务执行前的状态，释放占用的事务资源，并向协调者返回ack
  - 协调者收到所有参与者的ack消息，事务回滚完成

二、简答题

1. 什么是高内聚、低耦合?

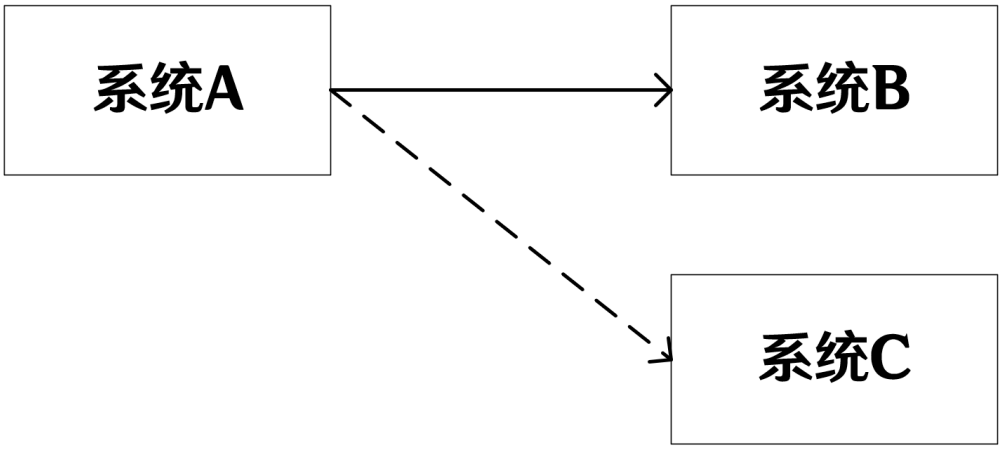
高内聚性：在面向对象编程中，若服务特型类型的方法在许多方面都很类似，则此类型即有高内聚性。在一个高内聚性的系统中，代码可读性及复用的可能性都会提高，程序虽然复杂，但可被管理。

低耦合性：组件之间的关系不强，使得一个组件的变更对其他组件的影响降到最低。每一个组件对其他独立组件的定义所知甚少或一无所知。

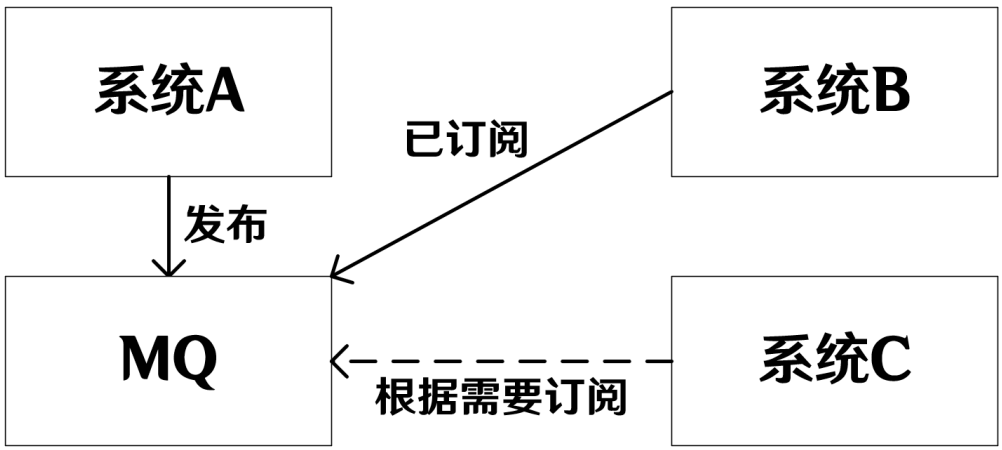
2. MQ(消息队列)如何解耦?

a. 没有使用MQ的情况

对于如上三个系统A、B、C，A需要发送数据给B，此时A与B直接耦合，但是如果有一天系统B不再需要系统A发送的消息，或如果系统C突然新增需求，需要接受系统A的消息，那么系统A需要频繁修改代码，系统耦合度高



b. 使用MQ的情况



系统A直接发布数据到MQ中间件，需要数据的系统直接订阅MQ即可，不需要数据则取消订阅。各系统间没有任何耦合度，以此达到解耦的目的。

3. 生产者(Producer)和消费者(Consumer)的关系

Producer和Consumer分别是消息队列中业务的发起方和处理方，Producer负责生产消息传输到Broker，而Consumer负责从Broker中获取消息并进行业务逻辑处理。

4. 计算机的计算架构的发展

- **C/S(Client-Server model)架构**：主从式架构，也称客户端/服务器架构，属于一种网络架构，它把客户端(Client，通常是一个采用图形用户界面的程序)与服务器(Server)区分开来。每一个客户端软件的实例都可以向一个服务器或应用程序服务器发出请求。
- **B/S(Browser/Server)架构**：与C/S结构不同，客户端不需要安装专门的软件，只需要浏览器即可，浏览器与Web服务器交互，Web服务器与后端数据库进行交互，可以方便地在不同平台下工作；服务器端可采用高性能计算机，并安装Oracle Database、DB2、MySQL等数据库。
- **从C/S到B/S的转变**：随着Internet和WWW的流行，以往的主机/终端和C/S架构都无法满足当前的全球网络开放、互连、信息随处可见和信息共享的新要求，于是就出现了B/S型模式。**它是C/S架构的一种改进，可以说属于三层C/S架构。**主要是利用了不断成熟的WWW浏览器技术，用通用浏览器就实现了原来需要复杂专用软件才能实现的强大功能，并节约了开发成本，是一种全新的软件系统构造技术。

5. 池(Pool)

a. 什么是池?

池可以想象为一个容器，其中保存着各种软件需要的对象。软件可以对这些对象进行复用，从而提高系统性能。

b. 池的分类:

- **资源池(对象池)**

  对象池(object pool pattern)是一种设计模式。一个对象池包含一组已经初始化过且可以使用的对象，而可以在有需求时创建和销毁对象。池的用户可以从池子中取得对象，对其进行操作处理，并在不需要时归还给池子而非直接销毁它。这是一种特殊的工厂对象。例如Socket连接池、JDBC连接池，CORBA对象池等等。

  优点：若初始化、实例化的代价高，且有需求需要经常实例化，但每次实例化的数量较少的情况下，使用对象池可以获得显著的效能提升。从池子中取得对象的时间是可预测的，但新建一个实例所需的时间是不确定。

- **线程池**

  线程池是一种线程使用模式。线程过多会带来调度开销，进而影响缓存局部性和整体性能。而线程池维护着多个线程，等待着监督管理者分配可并发执行的任务。

  优点：避免了在处理短时间任务时创建与销毁线程的代价。线程池不仅能够保证内核的充分利用，还能防止过分调度。

c. 对象池如何提高性能?

对象池将需要用到的对象存到池子中，需要用的时候取出(SetActive(true))，不需要的时候放回(SetActive(false))，在合适的时机将对象池清空，**省去了对象的频繁创建和销毁**，以此提高了性能。

d. 数据库连接池的作用?

数据库连接池是维护的数据库连接的缓存，以便在将来需要对数据库发出请求时可以重用连接。 连接池用于**提高在数据库上执行命令的性能**。为每个用户打开和维护数据库连接，尤其是对动态数据库驱动的网站应用程序发出的请求，既昂贵又浪费资源。**在连接池中，创建连接之后，将连接放在池中并再次使用，这样就不必创建新的连接。如果所有连接都正在使用，则创建一个新连接并将其添加到池中。**连接池还减少了用户必须等待创建与数据库的连接的时间。

6. 云原生

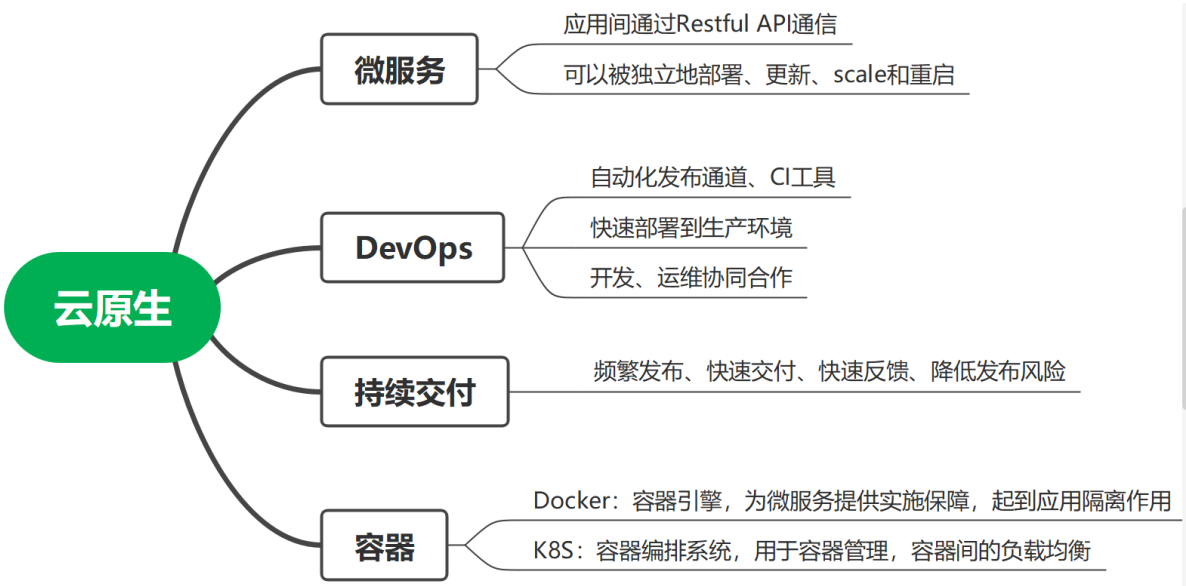
a. 什么是云原生？云原生的用途?

云原生(Cloud Native)是一种构建和运行应用程序的方法，有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式 API。

Cloud: 表示应用程序位于云中。

Native: 表示应用程序从设计之初即考虑到云的环境。

b. 云原生的4要素?



c. 云原生架构下的日志平台方案(不懂要怎么写)

7. 与单体应用架构相比，微服务架构有什么特点?

a. 传统架构存在的问题

- 项目过于臃肿，部署效率低下
- 开发成本高
- 无法灵活扩展

b. 微服务的特点

- **服务拆分粒度更细**

  微服务可以说是更细维度的服务化，小到一个子模块，只要该模块依赖的资源与其他模块都没有关系，那么就可以拆分为一个微服务。

- **服务独立部署**

  传统的单体架构是以整个系统为单位进行部署，而微服务则是以每一个独立组件为单位进行部署。

- **服务独立维护，分工明确**

  每个微服务都可以交由一个小团队进行开发，测试维护部署，并对整个生命周期负责，当我们将每个微服务都隔离为独立的运行单元之后，任何一个或者多个微服务的失败都将只影响自己或者少量其他微服务，而不会大面积地波及整个服务运行体系。

8. DNS负载均衡的优点和缺点

a. 优点

- 基本上无成本，因为往往域名注册商的DNS解析都是免费的。
- 部署方便，除了网络拓扑的简单扩增，新增的Web服务器只要增加一个公网IP即可。

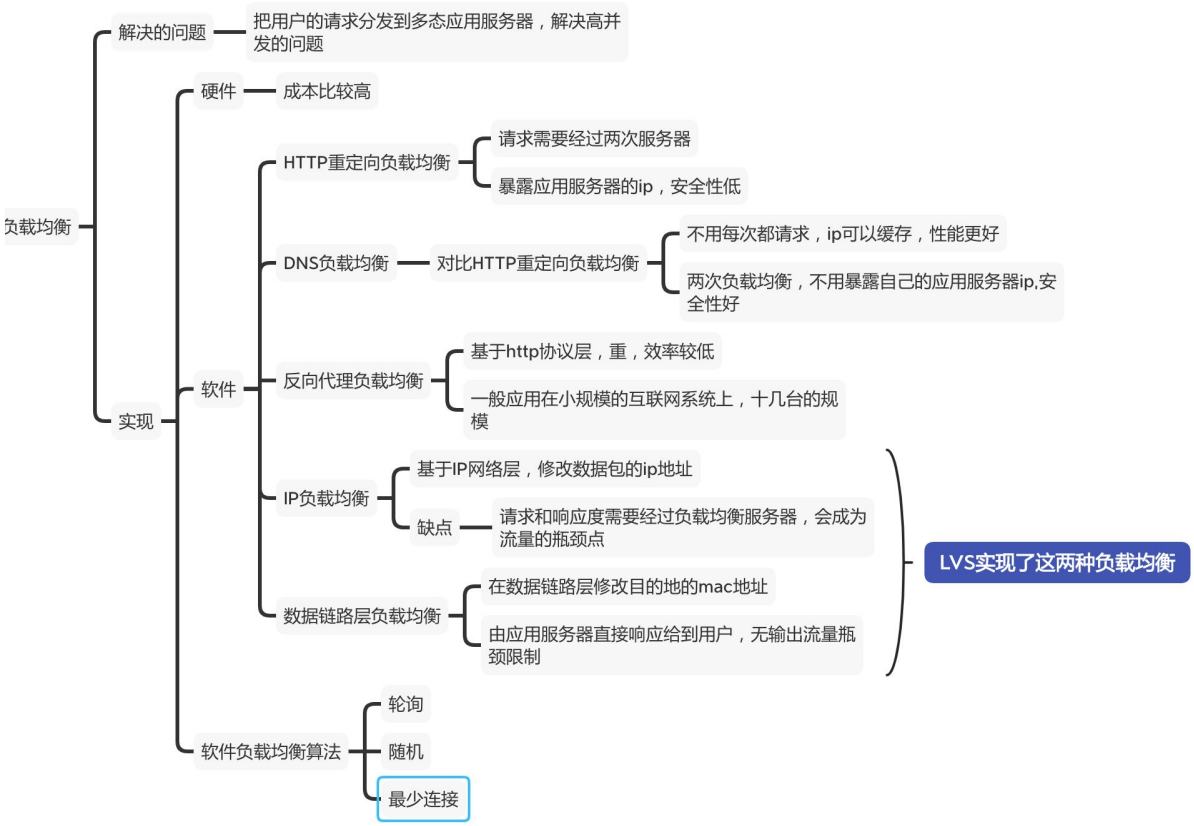
b. 缺点

- 健康检查，如果某台服务器宕机，DNS服务器是无法知晓的，仍旧会将访问分配到此服务器。修改DNS记录全部生效起码要3-4小时，甚至更久；
- 分配不均，如果几台Web服务器之间的配置不同，能够承受的压力也就不同，但是DNS解析分配的访问却是均匀分配的。其实DNS也是有分配算法的，可以根据当前连接较少的分配、可以设置Rate权重分配等等，只是目前绝大多数的DNS服务器都不支持；

- 会话保持，如果是需要身份验证的网站，在不修改软件构架的情况下，这点是比较致命的，因为DNS解析无法将验证用户的访问持久分配到同一服务器。虽然有一定的本地DNS缓存，但是很难保证在用户访问期间，本地DNS不过期，而重新查询服务器并指向新的服务器，那么原服务器保存的用户信息是无法被带到新服务器的，而且可能要求被重新认证身份，来回切换时间长了各台服务器都保存有用户不同的信息，对服务器资源也是一种浪费。

9. 如何实现DNS负载均衡

目前主流使用的两种是**DNS负载均衡**和**数据链路层负载均衡**



10. 序列化和反序列化

a. 序列化和反序列化的定义

序列化：把对象转化为可传输的字节序列的过程

反序列化：把字节序列还原为对象的过程

b. 为什么要序列化：

凡是需要**跨平台存储**和**网络传输**的数据，都需要序列化，序列化让对象可以跨平台存储和进行网络传输

c. 过程

网络传输本质：对象 → 字节序列 → 网络 → 字节序列 → 对象

跨平台存储本质：对象 → 跨平台字节码 → 跨平台 → 跨平台字节码 → 对象

11. 什么是Web服务？Web服务是无状态的吗？

Web服务是一种服务导向架构的技术，通过标准的Web协议提供服务，目的是保证不同平台的应用服务可以互操作。根据W3C的定义，**Web服务**应当是一个软件系统，用以支持网络间不同机器的互动操作。网络服务通常是许多API所组成的，它们通过Internet或远程服务端，执行客户所提交服务的请求。

存在基于状态和基于无状态的两种Web服务

- **在基于状态的Web服务中**，Client与Server交互的信息会保存在Server的Session中。再这样的前提下，Client中的用户请求只能被保存有此用户相关状态信息的服务器所接受和理解，这也就意味着在基于状态的Web系统中的Server无法对用户请求进行负载均衡等自由的调度。
- **在无状态的Web服务中**，每一个Web请求都必须是独立的，请求之间是完全分离的。Server没有保存Client的状态信息，所以Client发送的请求必须包含有能够让服务器理解请求的全部信息，包括自己的状态信息。使得一个Client的Web请求能够被任何可用的Server应答，从而将Web系统扩展到大量的Client中。

12. Rest风格的web服务可以是有状态的吗？请说明理由。

Rest风格的Web服务必须是**无状态**的。

13. 从技术商业角度阐述K8S产生并称为业界主流的原因？K8S和docker的关系?

> K8S称为业界主流的原因

- 技术角度

  > 技术背景

  容器技术之前，大家更多使用虚拟机(VM)进行开发，比如VMware和openstack，但是虚拟机对于开发和运维人员而言，存在启动慢，占用空间大，不易迁移的缺点。

  因此容器技术应运而生，它不需要虚拟整个操作系统，而只需要模拟一个小规模环境即可，启动速度快，基本不消耗额外的系统资源。Docker是应用最广泛的容器技术，通过打包镜像，启动容器来创建一个服务。

  但随着应用的逐渐复杂，容器的数量越来越多，由此衍生了管理运维容器的问题，于是K8S问世。

  > K8S实现了什么

  服务发现与调度，负载均衡，服务自愈，服务弹性扩容，横向扩容，存储卷挂载。总之，K8S可以使应用的部署和运维更加方便。

- 商业角度

  K8S在诞生之初就为云时代而生，拥有超前的眼光和先进的设计理念，加之最初由天才的 Google 工程师基于其内部 Borg 多年的经验设计而来，诞生之后就飞速发展。

  后来随着 RedHat、IBM、微软、Vmware、阿里云等来自全球的优秀工程师大力投入，打造了繁荣的社区和生态系统，成为企业容器编排系统的首选。

> K8S和docker的关系

- **Docker**是一个开源的**应用容器引擎**，开发者可以打包他们的应用及依赖到一个可移植的容器中，发布到流行的Linux机器上，也可实现虚拟化。
- **K8S(kubernetes)**是一个**开源的容器集群管理系统，可以实现容器集群的自动化部署、自动扩缩容、维护等功能。**

14. 面向切面编程(AOP)包含哪些要素?

- **Aspect**：切面的具体代码，即具体实现的功能
- **PointCut**：切面的执行位置(在哪个类的哪个方法执行)
- **Advice**：切面的执行时机(在目标方法前还是目标方法后等)

15. 什么是透明性，有几种透明性?

透明性是一种现象，即一个系统的分布情况对于用户和应用来说是隐藏的。包括：访问透明、位置透明、移植透明、重定位透明、复制透明、并发透明、故障透明和持久性透明。

16. 什么是Apache net，其优点和缺点是什么？(这题没听清楚题目，只听到优缺点)

- 优点：使得程序员能够以Java的方式编写服务器
- 缺点：运行依赖JDK、JRE

17. Spring Cloud

Spring Cloud是一个全家桶式技术栈，包含了很多组件。核心组件如下：

- **Eureka**

  微服务架构中的**注册中心**，专门负责服务的注册和发现。每个微服务中都有一个Eureka Client组件，该组件专门负责将这个服务的信息注册到Eureka Server中(告诉Eureka Server，自己在哪台机器上，监听了哪个端口)。而Eureka Server是一个注册中心，内置一个注册表，保存各服务所在的机器和端口号。

- **Feign**

  在需要**跨模块**发送请求的情况下，使用注解**@FeignClient**定义相应接口，Feign就会针对这一接口创建一个动态代理。后续调用接口，本质就是调用Feign创建的动态代理，该代理会根据接口上的**@RequestMapping**等注解，动态构造请求服务的地址。

- **Ribbon**

  Ribbon用于**负载均衡**，默认使用最经典的Round Bobin轮询算法

  > 假设服务A请求服务B，而服务B部署在5台服务器上，那么服务A会循环请求第1台、第2台、...、第五台

  Ribbon和Feign以及Eureka紧密协作，共同完成工作：

  - 首先Ribbon会从 **Eureka Client**里获取到对应的服务注册表，也就知道了所有的服务都部署在了哪些机器上，在监听哪些端口号。
  - 然后**Ribbon**就可以使用默认的Round Robin算法，从中选择一台机器
  - **Feign**就会针对这台机器，构造并发起请求。

- **Hystrix**：

  Hystrix是隔离、熔断以及降级的一个框架。Hystrix会管理很多微型的线程池，每个服务都拥有一个线程池，线程池里的线程仅仅用于请求对应服务。这样的好处是，如果有某个服务出了故障，其他服务的线程池还能正常工作，而不会受其影响。

  Hystrix还能对故障服务进行降级操作：每次调用故障服务，都会在数据库中保存操作日志，等待该服务修复完毕，即可根据数据库中的日志恢复正常结果。

- **Zuul**

  Zuul是微服务网关，负责网络路由。Zuul会收集前端发来的所有请求，根据请求的一些特征，将请求转发给后端的各个服务。

三、综合题

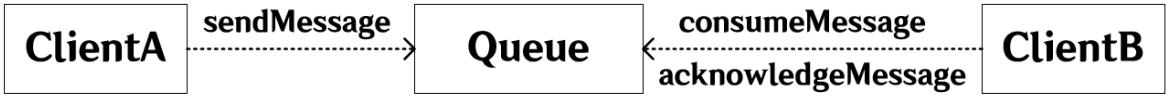
1. Queue/Topic模型

消息中间件一般有两种传递模式：点对点模式(P2P)和发布-订阅模式(Pub/Sub)。

- **P2P (Point to Point)** 点对点模型（Queue队列模型）
- **Publish/Subscribe(Pub/Sub)** 发布/订阅模型(Topic主题模型)

a. 点对点模型：生产者和消费者之间的信息往来

每个消息都被发送到特定的消息队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时。

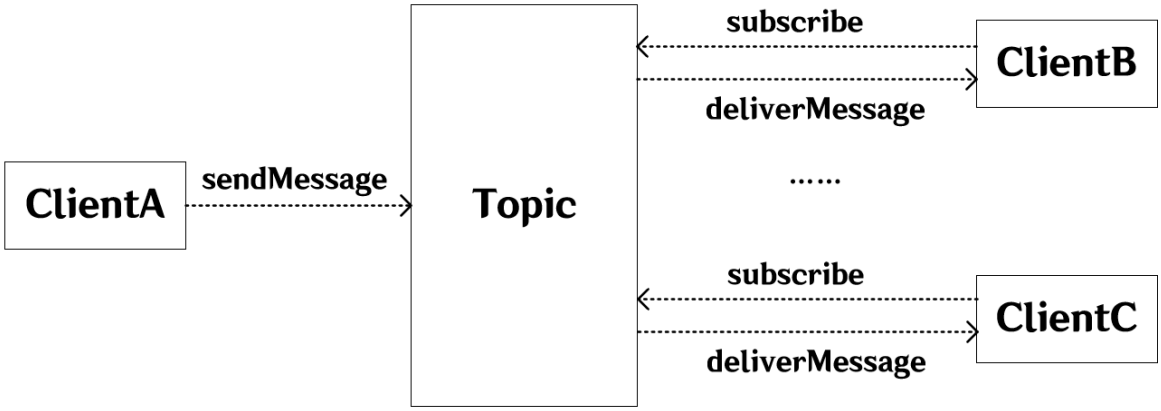


> 点对点模型的特点

- **每个消息只有一个消费者**（Consumer）(即一旦被消费，消息就不再在消息队列中)；
- **发送者和接收者之间在时间上没有依赖性**，也就是说当发送者发送了消息之后，不管接收者有没有正在运行，它不会影响到消息被发送到队列；
- **接收者在成功接收消息之后需向队列应答成功。**

b. 发布订阅模型

包含三个角色：主题(Topic)，发布者(Publisher)，订阅者(Subscriber)，多个发布者将消息发送到topic，系统将这些消息投递到订阅此topic的订阅者。



发布者发送到topic的消息，只有订阅了topic的订阅者才会收到消息。topic实现了发布和订阅，当你发布一个消息，所有订阅这个topic的服务都能得到这个消息，所以从1到N个订阅者都能得到这个消息的拷贝。

> 发布/订阅模型的特点

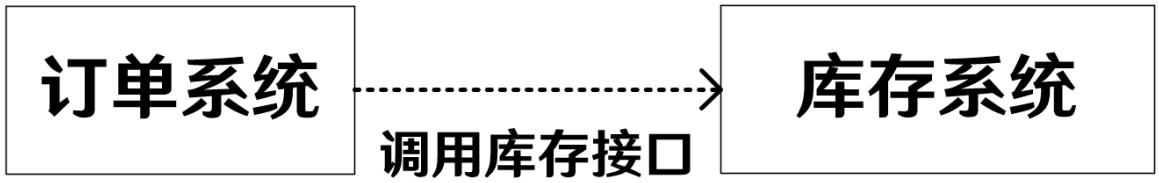
- **每个消息可以有多个消费者**
- **发布者和订阅者之间有时间上的依赖性**（先订阅主题，再来发送消息）
- **订阅者必须保持运行的状态，才能接受发布者发布的消息**

2. 如何使用中间件实现解耦、异步和削峰？(这里举例说明，具体场景具体分析)

a. 解耦

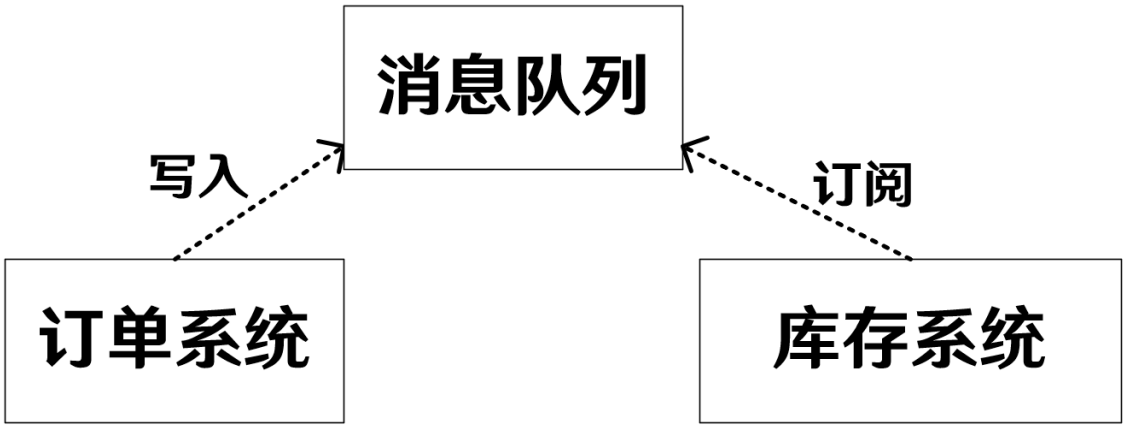
场景说明：用户下单后，订单系统需要通知库存系统。

> 传统做法：订单系统调用库存系统的接口



传统模式的缺点：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败，订单系统与库存系统耦合。

> 消息队列解决方案



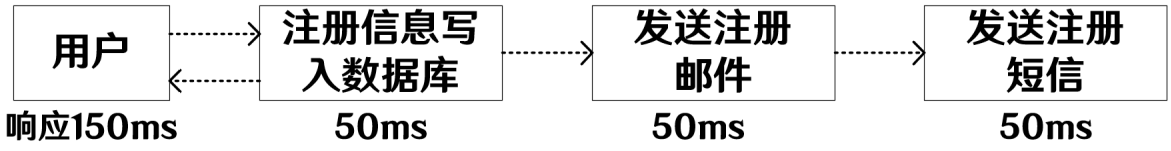
订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功 库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作 假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦。

b. 异步

场景说明：用户注册，需要执行三个业务逻辑，分别为写入用户表，发注册邮件以及注册短信。

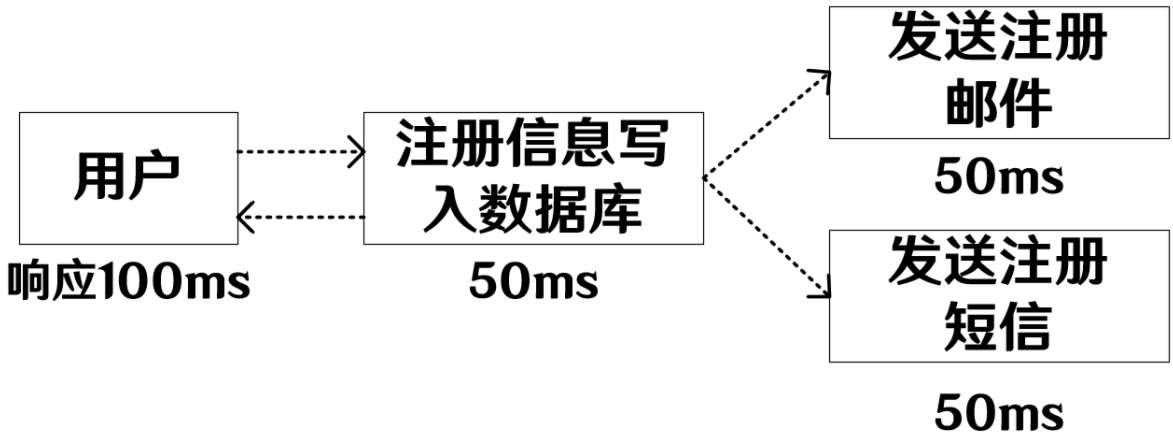
> 串行方式

将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。



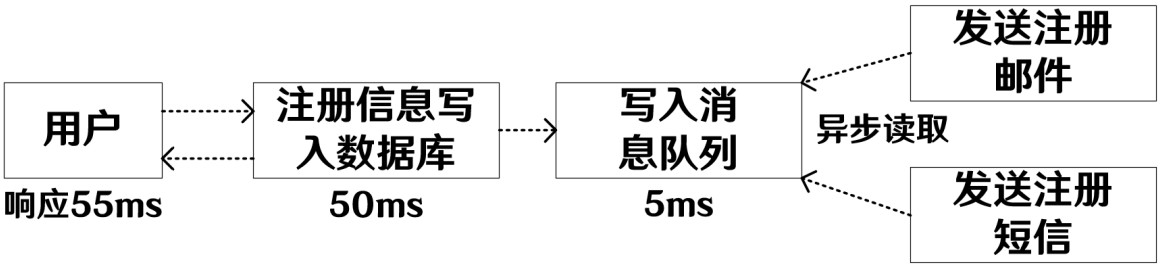
> 并行方式

将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间。



> 异步处理

引入消息中间件，将部分的业务逻辑，进行异步处理。改造后的架构如下：



c. 削峰

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面。秒杀业务根据消息队列中的请求信息，再做后续处理

3. CORBA为什么会失败?

a. 内部原因

- CORBA规模巨大，很难学习，开发过于复杂，这导致CORBA程序员的稀缺
- 商业CORBA费用昂贵，导致很多公司转向Web浏览器、Java和EJB的电子商务基础措施

b. 外部原因

- XML技术星期，SOAP使用XML作为RPC新的对象序列化机制
- IBM推出WebService的整套解决方案
- 基于JSON简单文本格式变化的HTTP REST通信方式的兴起

四、设计题(还没找)

1. 请你设计一个大规模在线订餐系统

a. 设计系统架构，画图整体架构图，解释其主要特征。

b. 若使用微服务，该如何对系统功能服务进行划分。(主要是纵向拆解)

c. 从中间件角度，谈谈短时间内大量访问的解决方法

2. 以12306网站订票系统为例，设计:

a. 设计系统架构

b. 说明技术特征

c. 谈谈短时间内大量访问的解决方法