题型

概念题 4分/道，共5道（总计20分）

简答题 5分/道，共8道（总计40分）

综合题 10分/道，共2道（总计20分）

设计题 20分/道，共1道（总计20分）

概念题考试范围（下划线为从其他文件中补充部分）

1. 中间件的概念

中间件是一种软件，处于**系统软件**（操作系统和网络软件）与**应用软件**之间，它能使应用软件之间进行**跨网络的协同工作**（也就是互操作），允许各应用软件之下所涉及的"系统结构、操作系统、通信协议、数据库和其它应用服务"各不相同。

> 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。

中间件的特性：

- **二进制级**的互操作
- 服务程序有多个客户（Client）
- 客户和服务都会升级进化，无约束，可组合性
- 对应用程序员，保持**编程模式的不变性**
- 客户和服务之间的契约既要具有永恒性，又具有可延伸性

2. 分布式互操作的概念

互操作是一种能力，使得分布的控制系统设备通过相关信息的数字交换，能够协调工作，从而达到一个共同的目标。传统上互操作是指 "**不同平台或编程语言之间交换和共享数据的能力**"。对于软件来说，互操作性是指不同厂商的设备，应该使用**通用的数据结构**和**传输标准设置**，使之可以互换数据和执行命令的解决方案。

互操作的范围：有三种形式，它们都要考虑**服务程序的标识**，**动态适配**

- **同一进程内不同模块**之间的互操作

  可直接完成

- **同一机器中不同进程**之间的互操作

  要求**进程中之间可以通信**，不用考虑**Data encoding**（数据编码）

- **不同机器之间模块**之间的互操作

  要求**机器间通信**，要考虑**不同Data encoding之间的转换**，因此须要中间翻译

  还要事先**约定传输协议**

3. 对象请求总线（ORB）的概念

ORB（Object Request Broker）是对象请求总线，又称对象请求代理。它允许**一台计算机通过网络对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**解释该调用**并负责**查找一个实现该请求的对象**，找到后，把参数传给该对象，调用它的方法，最后返回结果。

> 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。

补充：什么是负载均衡

负载均衡，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。

4. 容器的概念、容器与VM（虚拟机）的关系、为什么需要容器

容器概念：

CSDN：容器是**一组受到资源限制，彼此间相互隔离的进程。**

> 容器是**应用服务器**中位于**组件**和**平台**之间的接口集合。

阿里云：容器化是**应用程序级别的虚拟化**，允许单个内核上有多个独立的用户空间实例，这些实例称为容器。**容器**提供了将应用程序的代码、运行时、系统工具、系统库和配置打包到一个**实例**中的标准方法。容器共享一个**内核**（操作系统），它安装在硬件上。

VM概念：

> 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。

容器和虚拟机区别：

虚拟机和容器的结构

概要：虚拟机是基于硬体的多个客户操作系统，由虚拟机监视器（如VMware）实现。 容器是应用程序级构造，并模拟共享单个内核的多个虚拟环境。

具体区别：

	虚拟机	容器
系统性能	对于使用虚拟机的传统虚拟化，每个虚拟机都有自己的**完整操作系统**，因此在运行内置于虚拟机的应用程序时，内存使用量可能会高于必要值，虚拟机可能会开始耗尽主机所需的资源。	与传统的容器化应用程序不同，**共享操作系统环境**（内核），因此它们比完整虚拟机使用更少的资源，并减轻主机内存的压力。
占用	传统虚拟机可**占用大量磁盘空间**：除了虚拟机托管的任何应用程序外，它们还包含完整的操作系统和相关工具。	容器**相对较轻**：它们仅包含**使容器化应用程序运行所需的库和工具**，因此它们比虚拟机更紧凑，并且启动速度更快。
维护和更新	在更新或修补操作系统时，必须**逐个更新**传统计算机：必须**单独修补**每个客户操作系统。	对于容器，只需更新**容器主机**（托管容器的机器）的操作系统。这显著简化了维护。

> 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。**

为什么需要容器：

> 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。

5. 微服务架构（横向整合、纵向整合）的概念

微服务是近年来出现的一种新型的架构风格，它提倡将应用程序划分为许多**松散耦合且可独立部署**的较小组件或服务。与传统的服务架构相比，微服务架构更强调的是一种**独立测试、独立部署、独立运行**的软件架构模式。

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协议的**RESTful API**）。

每个微服务可以使用**不同的开发语言**、**不同的框架**、**不同的数据存储技术。**因此可以说，它是一种高度自治、非集中式管理、细粒度的业务单元。

（注：概念部分可联想OOAD的模块划分——每个模块都相当于一个服务，比如OrderService、UserServer；每个服务打包成一个jar包，部署在不同的服务器上；不同服务之间通过OpenFeign，并使用RESTful API进行跨模块调用）

> 横向整合：
>
> 在微服务架构中，每个微服务都应该围绕具体的业务进行构建，服务大小应该适中。在拆分微服务的过程中，应该尽量降低服务之间的耦合。服务拆分是为了横向扩展
>
> 纵向整合：
>
> 微服务纵向整合有三层：基础服务层（基础设施层）、组合服务器（平台服务层、支撑服务层、业务服务层）、控制层（网关层、接入层）

[注：简单来说，OOAD里的模块划分，比如划分为Order模块、User模块、Goods模块相当于横向整合；将OOAD整个系统纵向看，有最前端的网关、中间的业务层、后面的数据库层、底层的Ubuntu服务器，这就相当于纵向整合]

微服务的定义主要聚焦于四点：

- 小，且专注于一件事（业务独立、团队自主、高内聚低耦合）

- 运行在独立的进程中（每个服务都是一个具有高度自治性的独立业务个体）
- 轻量级的通信机制（语言无关、平台无关、代码无关，RESTful）
- 松耦合，独立部署

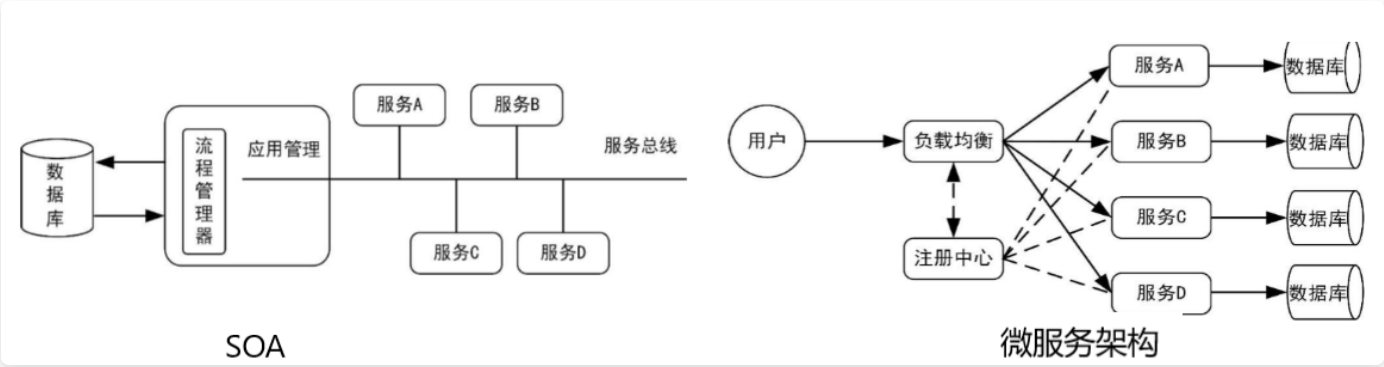
6. SOA的概念

SOA（Service-Oriented Architecture）面向服务架构是一个组件模型，它将应用程序的不同功能单元（称为服务）进行拆分，并通过这些服务之间定义良好的接口和协议联系起来。

SOA架构能够提高开发效率，降低系统之间的耦合，具有良好的扩展性。但SOA架构比较依赖**企业服务总线**（ESB），所有的服务都集中在一个 ESB 上。简单来说ESB就是一根管道，用来连接各个服务节点。

SOA和微服务架构的区别与联系：

- SOA和微服务架构是一个层面的东西（后者是前者的变体与升华），ESB和微服务网关是一个层面的东西
- 微服务**剔除SOA中复杂的ESB**，所有的业务智能逻辑在服务内部处理，使用轻量级的通信机制进行沟通
- SOA按**水平结构划分**：前端、后端、数据库、测试等；微服务强调按**垂直架构划分**，或者说按**业务划分**，每个服务完成一种特定的功能
- SOA倾向于使用**统一的技术平台**来解决所有问题；微服务可以**针对不同业务特征选择不同技术平台**，**去中心统一化**，发挥各种技术平台的特长



SOA和微服务架构

7. 组件的概念、面向组件编程的概念

~~组件是**对数据和方法的简单封装。**组件可以有自己的属性和方法，属性是组件数据的简单访问者，方法则是组件的一些可见的功能。~~（源自百度百科，个人觉得说的不好）

> 组件(Component)是对**数据和方法的简单封装。**

（注：**组件≠构件**，组件是Component，构件是Artifact）

对象管理小组的“建模语言规范”中将组件定义为：**系统中一种物理的、可代替的部件**，它封装了实现并提供了一系列可用的接口。一个组件代表一个系统中实现的物理部分，包括软件代码（源代码，二进制代码，可执行代码）或者一些类似内容，如脚本或者命令文件。

在C++ Builder中叫组件，在J2EE中叫Java Bean，在VB中叫控件。

面向组件编程，COP，Compment Oriented Programing [也有COD，面向组件设计的叫法]：

（注：这一部分在百度上没有明确的资料，主要参考小赖哥ppt的“组件技术”，考试写个大概就行）

组件技术是一种优秀的**软件重用技术**。采用组件开发软件就像搭积木一样容易，组件是具有某种特定功能的软件模型。

组件技术的基本思想是将复杂的大型系统中的**基础服务功能分解为若干个独立的单元**，即软件组件。

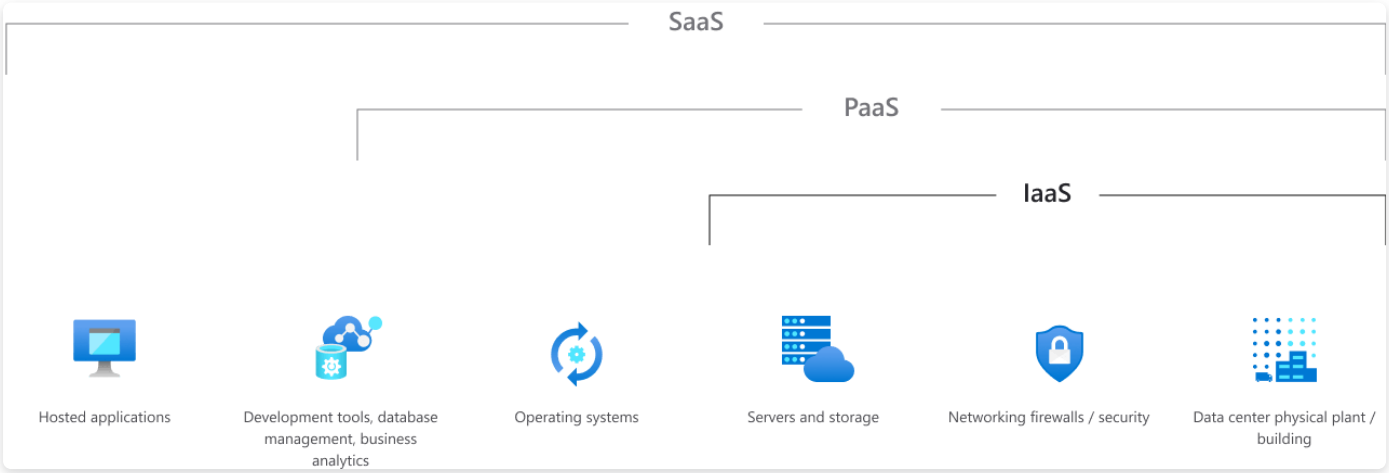
利用组件之间建立的**统一的、严格的连接标准**， 实现组件之间、组件与用户之间的服务连接。

组件技术将**面向对象特性**（例如封装和继承） 与（逻辑或物理的）**分布**结合起来，使面向对象技术进入到成熟的实用化阶段。

组件间的接口通过一种与平台无关的语言**IDL**（Interface Define Language）来定义。

> 面向组件编程，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AS的概念，如AAS、PAAS、IAAS、SAAS、CAAS、FAAS等（回答不用多，大概解释一下即可）



IAAS、SAAS、PAAS的关系

主要的三个AAS：

（AAS：**As a service**，作为服务）

- IAAS：**Infrastructure as a Service**，基础设施即服务，是云计算三种服务模式之一。它**把IT基础设施作为一种服务**通过网络对外提供，并根据用户对资源的实际使用量或占用量进行计费的一种服务模式。在这种服务模型中，普通用户不用自己构建一个数据中心等硬件设施，而是**通过租用的方式**，利用Internet从IaaS服务提供商获得计算机基础设施服务，包括服务器、存储和网络等服务。
- PAAS：**Platform as a Service**，平台即服务，是云计算三种服务模式之一。它为生成、测试和部署软件应用程序提供一个环境，或者说提供一个部署平台，抽象掉了硬件和操作系统细节，可以无缝地扩展。开发者只需要关注自己的业务逻辑，不需要关注底层。
- SAAS：**Software as a Service**，软件即服务，是云计算三种服务模式之一。它提供企业搭建信息化系统所需要的**所有网络基础设施及软件、硬件运作平台**，并**负责所有前期的实施、后期的维护等一系列服务**，企业无需购买软硬件、建设机房、招聘IT人员，即可通过互联网使用信息系统。

（打个比方，IAAS是租服务器，比如OOAD发的服务器；PAAS是租一个环境，比如华为云的数据库；SAAS是直接租一个成熟的软件或者系统，比如OOAD后期可以直接集成别的组的模块，或者说电脑订阅Office 365）

其他的AAS：

- BAAS：**Backend as a Service**，后端即服务。服务商为客户（开发者）**提供整合云后端**的服务，如提供文件存储、数据存储、推送服务、身份验证服务等功能，以帮助开发者快速开发应用。
- CAAS：**Communications as a Service**，通讯即服务。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（Software Development Kit，软件开发工具包）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。**
- DAAS：**Data as a Service**，数据即服务。通过**对数据资源的集中化管理**，并把**数据场景化**，为企业自身和其他企业的数据共享提供了一种新的方式。
- FAAS：**Function as a Service**，函数即服务。服务商提供一个平台，允许客户**开发、运行和管理应用程序功能，而无需构建和维护与应用程序启动、开发有关的基础架构。** 按照此模型构建应用程序是**实现"无服务器"体系结构**的一种方式，通常在构建微服务应用程序时使用。目前比较出名的有AWS的Lambada（简单来说，在写代码的时候直接调用这个服务）
- MAAS：**Mobility as a Service**，出行即服务。主要是**通过电子交互界面获取和管理交通相关服务**，以满足消费者的出行要求。通过对这一体系的有效利用，可充分了解和共享整个城市交通所能提供的资源，以此实现无缝对接、安全、舒适、便捷的出行服务。
- NAAS：**Network as a Service**，网络即服务。指客户可以**通过互联网访问第三方网络传输服务**，并采用**基于订阅模式的付费方式。**虚拟专用网（VPN）是使用最为广泛的NaaS产品之一。（比如，火箭加速器、白鲸加速器）

9. 英文单词缩写：RPC、LIOT（？）、IDL等的概念

可能会考到的英文单词缩写：

- RPC：**Remote Procedure Call**，**远程过程调用。**是**通过网络从远程计算机程序上请求服务**，而不需要了解底层网络技术。其目的是：让我们**调用远程方法像调用本地方法一样**无差别。

（注，LIOT没找到资料，可能说的是IIOT，或IOT，或IIOP）

- IIOT：**Industrial Internet of Things**，**工业物联网。**是将**具有感知、监控能力**的各类采集、控制传感器或控制器，以及移动通信、智能分析等技术不断**融入到工业生产过程各个环节**，从而大幅提高制造效率，改善产品质量，降低产品成本和资源消耗，最终实现**将传统工业提升到智能化**的新阶段。
- IOT：**Internet of Things**，**物联网。**是互联网基础上的延伸和扩展的网络，将**各种信息传感设备与网络结合起来**而形成的一个巨大网络，**实现任何时间、任何地点，人、机、物的互联互通。**
- IDL：**Interface Definition Language**，**接口定义语言。**它是CORBA规范的一部分，是用于**描述分布式对象接口的定义语言。**IDL用**中立语言的方式**进行描述，能使**软件组件（不同语言编写的）间相互通信。**使用IDL进行接口定义后，就确定了客户端和服务器之间的接口，这样即使客户端和服务器独立进行开发，也能够正确的定义和调用所需要的分布式方法。
- DDL：**Distributed Description Logic，分布式描述逻辑。**它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，相互之间使用桥规则（Bridge Rule）进行连接。
- DLL：**Dynamic Link Library**，**动态链接库。**它又称"应用程序拓展"，是一种软件文件类型。在Windows中，许多应用程序并不是一个完整的可执行文件，它们**被分割成一些相对独立的动态链接库**，即DLL文件，放置于系统中。当我们执行某一个程序时，相应的DLL文件就会被调用。一个应用程序可使用多个DLL文件，一个DLL文件也可能被不同的应用程序使用，这样的DLL文件被称为共享DLL文件。
- CORBA：**Common Object Request Broker Architecture**，**公共对象请求代理体系结构。**它是由 OMG（对象管理组织，Object Management Group）提出的应用软件体系结构和对象技术规范。其核心是**一套标准的语言、接口和协议**，以支持**异构分布应用程序间的互操作性**及**独立于平台和编程语言的对象重用。**
- 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，提供了一个标准传输语法，用于在ORB之间进行通信。GIOP**只能用在ORB与ORB之间**，而且，只能在符合理想条件的面向连接传输协议中使用。
- IIOP：**Internet Inter-ORB Protocol，互联网内部对象请求代理协议。**它是一个**实现互操作性**的协议，是CORBA中至关重要的一部分。它使得由不同语言编写的分布式程序**在因特网中**可以实现彼此的交流沟通。

- DNS：**Domain Name System，域名系统。**是**互联网的一项服务。**它作为**将域名和IP地址相互映射**的一个分布式数据库，能够使人更方便地访问互联网。
- C/S：Client-Server，客户端/服务端模式。是一种架构，服务器负责**数据的管理**，客户机负责**完成与用户的交互任务。**
- B/S：Browser/Server，浏览器/服务器模式。是WEB兴起后的一种**网络结构模式**，WEB浏览器是客户端最主要的应用软件。这种模式**统一了客户端**，将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用。
- TCP：**Transmission Control Protocol**，**传输控制协议。**是一种**面向连接的**、**可靠的**、**基于字节流**的传输层通信协议。
- UDP：**User Datagram Protocol，用户数据报协议。**一种**无连接**的传输层协议，提供面向事务的简单的、**不可靠的**信息传送服务。

（其余的一些知识点，比如SOA、ORB等如果没有单独考察，可能会放在英文缩写里考）

10. 两段式提交、两段式锁的概念

10.1 两段式提交（此处仅介绍概念，详细的在简答题中）

Two Phase Commit，2PC，两段提交协议。

概念：2PC是**将整个事务流程分为两个阶段**——**准备阶段**（Prepare phase）、**提交阶段**（Commit phase）。

角色：在两阶段提交的过程中，主要分为了**协调者（Coordinator）**和**参与者（Participant）**两种角色。协调者主要就是起到协调参与者是否需要提交事务或者中止事务，参与者主要就是接受协调者的响应并回复协调者是否能够参与事务提交。

过程：

- 准备阶段：事务管理器给每个参与者发送Prepare消息，每个数据库参与者在本地执行事务，并写本地的Undo/Redo日志，此时事务没有提交。
- 提交阶段：如果事务管理器收到了参与者的执行失败或者超时消息时，直接给每个参与者发送回滚(Rollback)消息；否则，发送提交(Commit)消息；参与者根据事务管理器的指令执行提交或者回滚操作，并释放事务处理过程中使用的锁资源。注意:必须在最后阶段释放锁资源。

10.2 两阶段锁

Two Phase Lock，2PL，又称两段锁。

概念：两阶段锁指事务必须分两个阶段对数据进行**加锁（Growing Phase，增长阶段、生长阶段）**、**解锁（Shrinking Phase，缩减阶段、衰退阶段）**这两项操作，每项操作只能在一个阶段内进行。**在同一个事务内，加锁和解锁操作不能交叉执行。**也就是说，在加锁阶段，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；在解锁阶段，事务可以释放任何数据项上的任何类型的锁，但不能加锁。

[注意，两阶段锁 ≠ 两类锁，为了提高并发度，才对锁进行分类，分出**共享锁（读锁）和排它锁（写锁）**]

简答题考试范围（下划线为从其他文件中补充部分）

1. 高内聚低耦合的概念
    - 主要在于解耦，如生产者消费者如何解耦、命名时名字和地址怎么解耦
    - 消息性中间件是降低耦合的一个重要工具，它是如何帮助我们解耦的，<u>如：怎么把多对多变成的一对一、一对多（模型、producer、consumer等等）</u>
2. 计算架构，演进顺序：单击——C/S、B/S——分布式——云计算——端管边云融合，其背后驱动力有：

    - 技术驱动力，主要是计算的分配、计算的要求
    - 商业驱动力，主要是单位计算资源越来越便宜，如果再加一句，就是谁掏钱的问题，是由Service Provider（消息提供者）来掏钱，还是由Consumer（客户）来掏这些计算资源的钱
3. 池，比如对象池、数据库连接池，利用对象池如何提高性能，提高性能的原理是什么，高星简略地说了两点：
    - 降低资源建立和退出的开销
    - 对建立的资源实现有效的高效并发调度
4. 云原生的概念，以日志系统来解释云原生的技术特点，尤其是它相对于传统技术的优点；内核、理解、往外抛概念、4个特点（注：云原生的概念需要自己搜一些资料，因为上课没讲太细，必考，写个一两行字即可）
5. 两阶段提交，图必考

   两阶段提交按照时间轴会出现特殊情况，这些特殊情况（即失败情况）怎么处理

   需要能把这些特殊情况按照时间轴列出来，并说说该怎么处理
6. 微服务架构：微服务架构的定义、特点、优势

   如果用一句话概括微服务架构和SOA——微服务架构有利于进行纵向整合、SOA有利于进行横向整合

   （答题不能写这么简略，最好自己再多找一些资料）
7. 负载均衡：（高星说负载均衡必考）
    - 负载均衡有多种方式，如轮询、根据资源使用情况、根据网络情况
    - DNS负载均衡：如何实现？属于第几层负载均衡？原理是什么？有什么优点和缺点？
8. 对象的序列化和反序列化：需要知道概念、作用、过程
9. 面向切面的编程（AOP）：包含哪些要素
10. 透明性：需要解释概念、有几种透明性（比如位置透明性、迁移透明性），大概总共有6-7个
11. Apalet：概念、优缺点，比如优点是可以用Java语言编写客户端程序，缺点是Apalet依赖于JDK库（或JRE），在微软不再在IDE集成JDK后，Apalet就不能用了

    与Apalet对比：Servlet的概念、优缺点
12. SpringCloud：包含哪些特性、有哪些核心组件，简要介绍核心组件运行流程
13. 数据库连接池（上面的3是对象池）：池化的作用

综合题考试范围（下划线为从其他文件中补充部分）

1. 消息中间件中：Topic和Queue模型、如何用消息中间件实现解耦、异步、削峰（同简答题1）
2. CORBA如此强大，为何在推广上会失败

   说说你对跨平台、跨语言的技术框架未来发展的看法
3. 什么是Web服务，它是有状态的还是无状态的

   REST风格的Web服务可以有具体状态吗？给出答案并解释原因

   （这道题不一定是综合题，但必然以某种方式出现）
4. 从技术和商业角度，阐述k8s（Kubernetes）产生以及成为业界主流的原因，说明其主要特征，解释其与docker的关系

开发设计题（下划线为从其他文件中补充部分）

1. 以大规模在线订餐为例，设计系统的架构、画出整体架构图、解释其主要特点

   如果是基于微服务来设计，该如何对系统的功能和服务进行划分

   从中间件的角度谈谈，可以从哪些层次、哪些技术方案着手，以应对短时间内大量访问的应用需求（即高并发大负载）

2. 以12306为例，设计系统的功能结构、讨论可以用哪些技术、框架可以应对短时间内大量访问的应用需求（第二题类似第一题）