简答题

高内聚低耦合

- 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，如：怎么把多对多变成的一对一、一对多（模型、producer、consumer等等）

概念：

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。

生产者消费者如何解耦：

生产者消费者模式是通过一个容器来解决生产者和消费者的强耦合问题。生产者和消费者彼此之间不直接通讯，而通过阻塞队列来进行通讯，所以生产者生产完数据之后不用等待消费者处理，直接扔给阻塞队列，消费者不找生产者要数据，而是直接从阻塞队列里取，阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力。

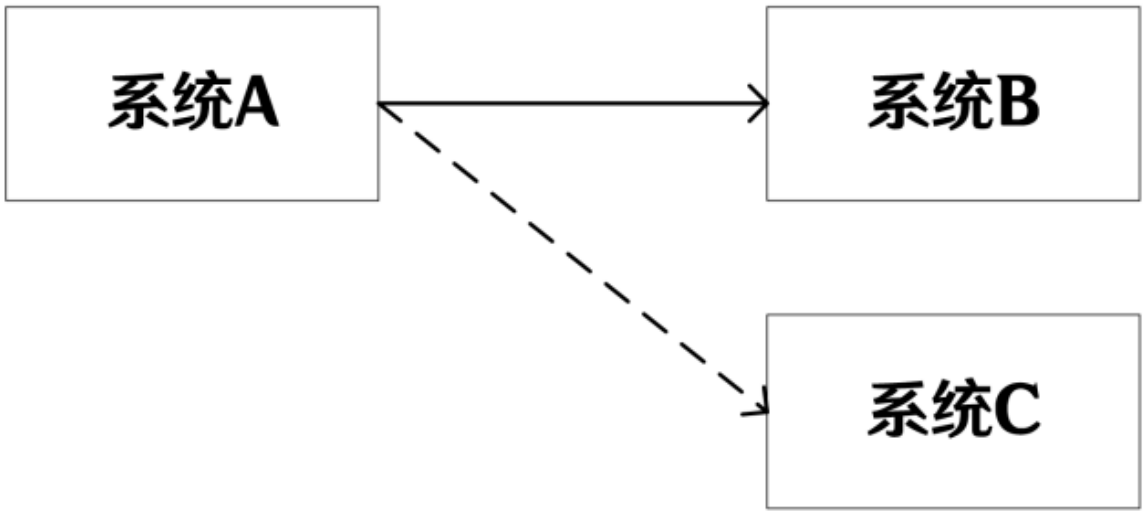
命名时名字和地址如何解耦：

(来自炫哥PPT：1647769367412第二章 分布式处理环境)

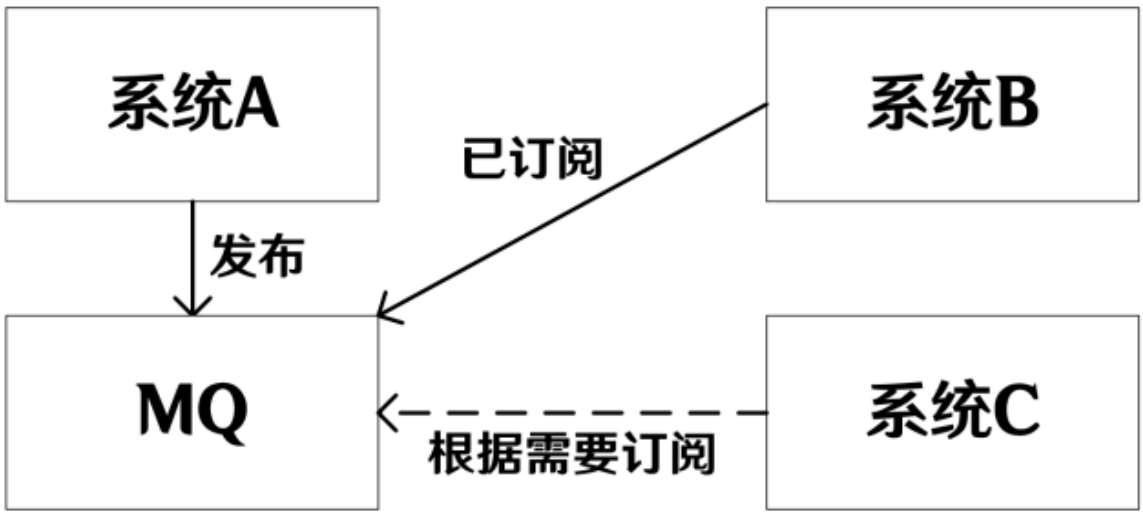
命名时通过引入一个目录服务器如CDS(Cell Directory Service)，来实现名字和地址的解耦。正在初始化的应用服务器将该服务器的主机地址及每个界面的名字写入CDS服务器进行服务注册，这样客户端使用服务名就能通过CDS服务器进行查找，调用到对应的服务，服务名字与地址的修改相互独立，实现名字与地址解耦。

消息中间件是如何解耦的：

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。

计算架构

演进顺序：单机——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

计算架构概述：

- **单机**：单机就是所有业务全部写在一个项目中，部署服务到一台服务器上，所有请求业务都由这台服务器处理，显示，当业务增长到一定程度的时候，服务器的硬件会无法满足业务需求。
- **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。

- **分布式计算架构**：分布式计算是把一个需要非常巨大的计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给多个计算机进行处理，最后把这些计算结果综合起来得到最终的结果。实际应用场景中，分布式结构就是一个完整的系统，按照业务功能，拆分成一个个独立的子系统，在分布式结构中，每个子系统就被称为“服务”。这些子系统能够独立运行在Web容器中，他们之间通过RPC方式通信。
- **云计算**：云计算是一种商业计算模型。它将计算机任务分布在大量计算机构成的资源池上，使各种应用系统能够根据需要获取计算能力、存储空间和信息服务。简要来说：云计算是通过网络按需提供可动态伸缩的廉价计算服务”，这种资源池称之为“云”。
- **端管边云**：所谓的“云”是云计算平台，“管”则是指有线/无线通讯方式，“边”是指边缘计算，“端”则涵盖了智能传感、智能终端和智能设备。(边缘计算，是一种分散式运算的架构，将应用程序、数据资

料与服务的运算，由网络中心节点，移往网络逻辑上的边缘节点来处理。）

计算架构演进的驱动力：

1. 人们需要获得更高的计算性能。云计算、分布式可以解决大数据导致的计算量急剧增长和弹性要求，（大量的计算机意味着）大量的并行运算，大量CPU、大量内存、以及大量磁盘在并行的运行。
2. 云计算满足人们随时随地接入的使用习惯，且由Consumer（客户）来支付这些租用计算资源的钱；
3. 云计算带来新增的赢利点，采购规模效应使得单位计算资源越来越便宜，并大幅降低成本
4. 另一个人们构建分布式系统、云计算的原因是，它可以提供容错（tolerate faults）。比如两台计算机运行完全相同的任务，其中一台发生故障，可以切换到另一台。
5. 第三个原因是，一些问题天然在空间上是分布的。例如银行转账，我们假设银行A在纽约有一台服务器，银行B在伦敦有一台服务器，这就需要一种两者之间协调的方法。所以，有一些天然的原因导致系统是物理分布的。
6. 人们构建分布式系统来达成一些安全的目标。将代码分散在多处运行，通过一些特定的网络协议通信，这样可以限制出错域。

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

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，而且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。

炫哥PPT(1653910245171第九章%20池化和负载均衡中间件 第5页)：对象池技术的核心是缓存和共享，即对于那些被频繁使用的对象，在使用完后不立即将它们释放，而是缓存起来。这样后续的应用程序可以重复使用这些对象，从而减少创建对象和释放对象的次数，改善应用程序的性能。

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。

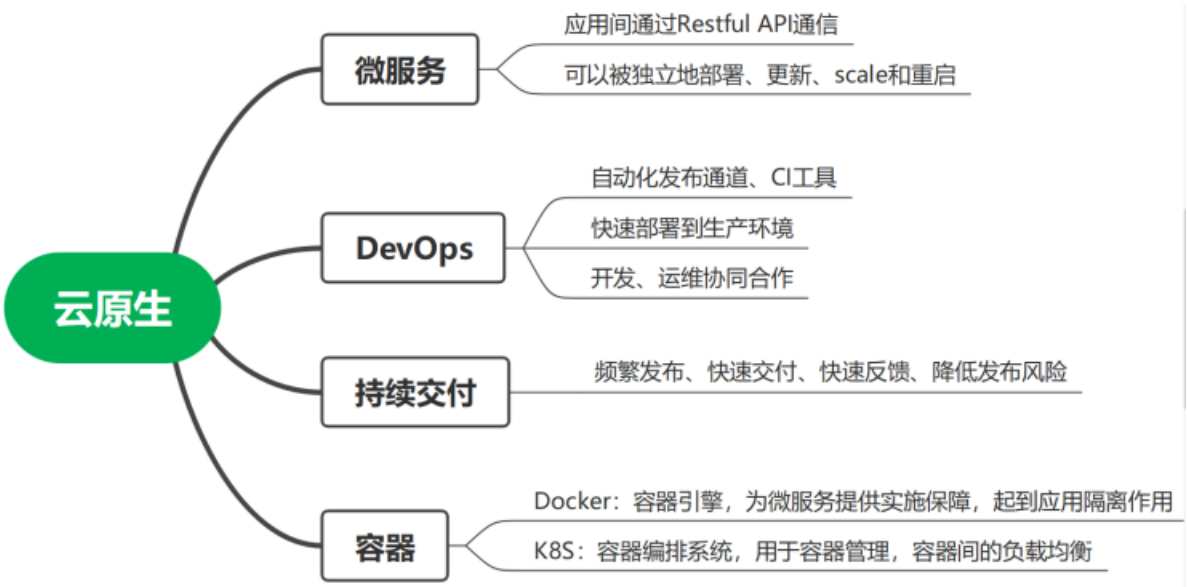
云原生的概念

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）

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. 以日志系统来解释云原生的技术特点

云原生环境下的日志系统与传统系统相比有以下特点：

高可用：基于云原生的微服务架构，可实现存算分离，所有服务都是无状态的，故障快速恢复。
高性能：所有集群都可横向扩展，没有热点。
分层设计：各模块之间低耦合，模块之间定义标准接口，组件可替换

传统方式下，日志管理相对简单。日志的数量，类型和结构都很简单且易于管理。但在云原生环境下，日志系统与传统的日志方式有很大差别，主要有以下几点：

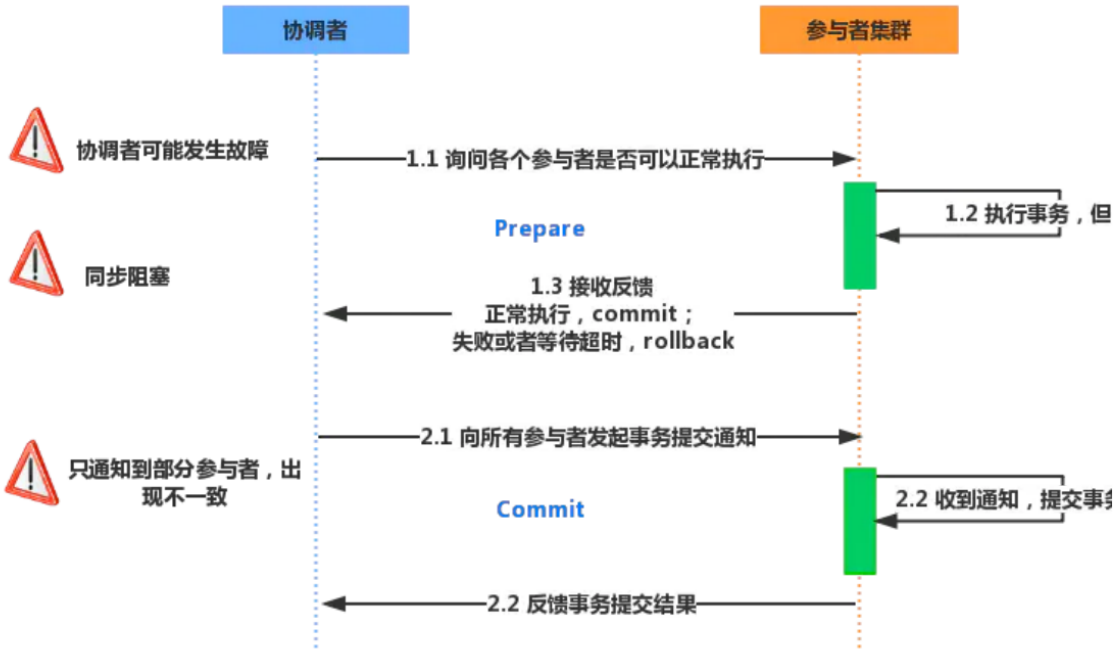
1. 在云原生时代之前，大多数应用程序都是运行在单个服务器上的整体组件。每个服务器通常还只生成少量日志。相比之下，在云原生环境中，通常使用微服务体系结构-可能有十几个或更多不同的服务在运行，每个服务都提供了组成整个应用程序所需的不同功能。每个微服务都可以生成自己的日志。
2. 不仅如此，总体上不仅有更多的日志，而且还有更多类型的日志。云原生环境下不仅产生服务器日志和应用程序日志，还有云基础架构的日志，Kubernetes或Docker的日志，身份验证日志等等。
3. 当容器实例停止运行时，存储在容器中的所有数据将被永久销毁。因此，如果日志数据存储在容器内（默认情况下通常是默认情况下），它将与容器一起消失。

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

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

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）

(炫哥PPT：1653910265671第十一章%20微服务)

定义：（P4)

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一组细粒度的服务，服务间采用轻量级的通信机制进行交互。 在微服务架构中，每个微服务都是具有单一职责的小程序，能够被独立地部署、扩展和测试。通过将这些独立的服务进行组合可以完成一些复杂的业务。

特点：

微服务具有以下特点：(P13)
✓ 1) 微服务之间是松耦合的。微服务的功能可分为业务功能和技术功能，各服务之间耦合度较低。每个微服务都是单一职责的。
✓ 2) 各服务之间是独立部署的。当改变一个特定的微服务时，只需要将该微服务的变更部署到生产环境中，而无需部署或触及系统的其他部分。
✓ 3) 每个微服务有独立的数据存储，易于维护。采用独立的数据存储，能有效避免微服务间在数据库层面的耦合。可以根据微服务的业务和需求选择合适的数据库技术。

SOA与微服务的区别：(P10)

微服务是 SOA 的子集。微服务架构与SOA架构主要有以下3点区别：
✓ 1) 架构划分不同。SOA强调按水平架构划分。微服务强调按垂直架构划分，按业务能力划分。
✓ 2) 技术平台选择不同。SOA应用倾向于使用统一的技术平台。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。
✓ 3) 系统间边界处理机制不同。SOA架构强调的是异构系统之间的通信和解耦合。微服务架构强调的是系统按业务边界做细粒度的拆分和部署。

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？

炫哥PPT:1653910245171第九章%20池化和负载均衡中间件

实现方式：(P75)

负载均衡技术在实现方式中，主要是在应用层（7层）、传输层（4层）、网络层（3层）等做文章。
网络层（3层）：当负载均衡服务器接受到请求之后，根据不同的负载均衡算法，通过IP将请求转发至不同的真实服务器。
传输层（4层）：四层负载均衡服务器在接受到客户端请求后，通过修改数据包的地址信息（IP+端口号）将流量转发到应用服务器。代表性产品是LVS，F5。
应用层（7层）：除了根据IP加端口进行负载外，还可根据7层的URL、浏览器类别、语言来决定是否要进行负载均衡。代表性产品nginx（软件）、apache（软件）

负载均衡调度算法：(89)

轮叫调度、加权轮叫调度、最少链接调度、加权最少链接调度

DNS负载均衡：

DNS负载均衡技术是最早的负载均衡解决方案。属于第七层。它通过 DNS服务中的随机名字解析来实现的。在DNS服务器中，可为同一个域名配置多个不同的地址。查询域名的客 户机可获得其中的一个地址。因此对于同一个域名，不 同的客户机会得到不同的地址，并访问不同地址上的 Web服务器，达到负载均衡的目的。

优缺点：

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。
缺点：
可能存在以下的问题：
✓ 1) 负载分配不均匀：未考虑每个Web服务器当前的负 载情况，最慢的Web服务器将成为系统的瓶颈
✓ 2) 可靠性低：如果某台Web服务器出现故障，DNS 服务器仍然会把请求分配到这台故障服务器上，导致 不能响应客户端；
✓ 3) 变更生效时间长：更改DNS的配置时，有可能造成 相当一部分客户不能使用Web服务；并且由于DNS缓 存的原因，所造成的后果要持续相当长一段时间。

对象的序列化和反序列化：

需要知道概念、作用、过程

- 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

面向切面的编程（AOP）：

AOP即为Aspect Oriented Programming的缩 写，意为面向切面的编程，是通过预编译方式 和运行期动态代理的方式，实现程序功能的统 一维护的一种技术。

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)

透明性：

炫哥PPT(1647769424788第三章%20%20ODP和Corba介绍.pdf)
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

透明性概念：

屏蔽了由系统的分布所带来的复杂性:ODP(分布式处理Open Distributed Processing)易用性的基础是程序员和最终用户不需要
关注分配,换句话说，编程和使用分布式应用程序看起来完全一样,就好像应用程序根本没有分发一样。

透明性包括：

访问透明性 access transparency：用相同的操作访问本地资源和远程资源。
位置透明性 location transparency：不需要知道资源的物理或网络位置就能访问它们
迁移透明性 migration transparency：资源和客户能够在系统内移动而不会影响用户或程序的操作
失败透明性 failure transparency：屏蔽错误，不论是硬件组件故障还是软件组件故障，用户和应用程序员能够完成他们的任务
重定位透明性 relocation transparency：用户不必知道资源的位置是否改变
复制透明性 replication transparency：使用资源的多个实例提升可靠性和性能，而用户和应用程序员无需知道副本的相关信息
持久透明性 persistence transparency：对象所处的状态既可以是活动的，也可以是静止的。
事务处理透明性 transaction transparency：与事务处理相关的调度、监控和恢复对用户透明

Applet：

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Applet依赖于JDK库（或JRE），在微软不再在IDE集成JDK后，Applet就不能用了，与Applet对比：Servlet的概念、优缺点

概念：

Applet 是一种 Java 程序。它一般运行在支持 Java 的 Web 浏览器内。因为它有完整的 Java API支持,所以Applet 是一个全功能的 Java 应用程序。

优点：

可以用Java语言编写客户端程序，
applet运行无需安装，
安全：不能运行本地的可执行文件；不能读取或编写本地计算机文件系统；

缺点：

Applet依赖于JDK库（或JRE），在微软不再在IDE集成JDK后，Applet就不能用了
applet启动速度慢
JVM装载时间慢

Servlet：

概念：

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，具有独立于平台和协议的特性，主要功能在于交互式地浏览和生成数据，生成动态Web内容。

与 Applet 的比较

相似之处：

- 它们不是独立的应用程序，没有 main() 方法。
- 它们不是由用户或程序员调用，而是由另外一个应用程序(容器)调用。

- 它们都有一个生存周期，包含 init() 和 destroy() 方法。

  不同之处：

- Applet具有很好的图形界面(AWT)，与浏览器一起，在客户端运行。
- Servlet 则没有图形界面，运行在服务器端。

servlet优缺点：
Servlet优点：

1、 是mvc的基础，其他的框架比如struts1，struts2，webwork都是从servlet基础上发展过来的。所以掌握servlet是掌握mvc的关键。
2、 Servlet把最底层的api暴漏给程序员，使程序员更能清楚的了解mvc的各个特点。
3、 程序员可以对servlet进行封装。Struts2就是从servlet中封装以后得到的结果。
4、 市场上任何一个mvc的框架都是servlet发展过来的，所以要想学好struts2这个框架，了解servlet的运行机制很关键。
Servlet的缺点
1、 每写一个servlet在web.xml中都要做相应的配置。如果有多很servlet，会导致web.xml内容过于繁多。
2、 这样的结构不利于分组开发。
3、 在servlet中，doGet方法和doPost方法有HttpServletRequest和HttpServletResponse参数。这两个参数与容器相关，如果想在servlet中作单元测试，则必须初始化这两个参数。(具有容器依赖性)
4、 如果一个servlet中有很多个方法，则必须采用传递参数的形式，分解到每一个方法中。

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

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。

数据库连接池（上面的3是对象池）：池化的作用

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。

池化的作用：

在高并发环境下，程序涉及到大量系统调用，消耗大量的 CPU资源，频繁申请释放小块内存的部分代码常常成为整 个程序的性能瓶颈。池化技术能够减少资源对象的创建次数，提高程序的性能，包括对象池技术数据库连接池技术 线程池技术等。