# 目录结构

# 相关说明

# 机构声明

2019年每特教育独创Java微服务架构面试宝典,针对三年以上工作经验跳槽使用，涵盖微服务电商项目架构、互联网安全平台架构、SpringCloud/SpringBoot、分布式解决方案、消息中间件、实战高并发解决方案、数据库/JVM/Tomcat性能优化、网络通讯Netty/Nio、数据结构与算法等。

# 版本更新

# 微服务架构相关

## 大型网站架构演变过程

### 网站架构演变演变过程

传统架构 → 分布式架构 → SOA架构 → 微服务架构

### 什么是分布式架构

分布式架构就是将传统结构按照模块进行拆分，不同的人负责不同的模块，不会产生代码冲突问题，方便开发。

### 什么是SOA架构

SOA架构就是将业务逻辑层提取出来，将相似的业务逻辑形成一个服务，提供外部访问接口，服务之间访问通过RPC调用实现。

### 什么是微服务架构

微服务类似于SOA架构，但是比SOA架构粒度更细，更轻量。

### 微服务架构与SOA架构区别

SOA基于WebService和ESP实现，底层基于HTTP协议和使用XML方式传输，XML在网络传输过程中会产生大量冗余。微服务由SOA架构演变而来，继承了SOA架构的优点，同时对SOA架构缺点进行改善，数据传输采用JSON格式，相比于XML更轻量和快捷，粒度更细，更加便于敏捷开发。SOA数据库会存在共享，微服务提倡每个服务连接独立的数据库。

## 微服务框架SpringCloud

### 什么是SpringCloud

SpringCloud是微服务的一种解决方案，依赖SpringBoot实现。包含注册中心(eureka)、客户端负载均衡(Ribbon)、网关(zull)、分布式锁、分布式会话等。

### 为什么要使用SpringCloud

SpringCloud是一套非常完整的微服务解决方案，俗称“微服务全家桶”，几乎内置了微服务所使用的各种技术，可以不必集成第三方依赖。

### SpringCloud服务注册发现原理

每个SpringCloud服务器启动后向注册中心注册本服务器信息，如服务别名、服务器IP、端口号等，其他服务进行请求时先根据服务别名从注册中心获取到目标服务器IP和端口号，并将获取到的信息缓存到本地，然后通过本地使用HttpClient等技术进行远程调用。

### SpringCloud 支持那些注册中心

Eureka、Consul、Zookeeper

### 现在Eureka闭源了，可以通过什么注册中心替代Eureka呢？

Consul或Zookeeper

### 谈谈你对微服务服务治理的思想

### Eureka如何实现高可用

启动多台Eureka服务器，然后作为SpringCloud服务互相注册，客户端从Eureka集群获取信息时，按照注册的Eureka顺序对第一个Eureka进行访问。

### @LoadBalanced注解的作用

开启客户端负载均衡。

### Nginx与Ribbon的区别

Nginx是反向代理同时可以实现负载均衡，nginx拦截客户端请求采用负载均衡策略根据upstream配置进行转发，相当于请求通过nginx服务器进行转发。Ribbon是客户端负载均衡，从注册中心读取目标服务器信息，然后客户端采用轮询策略对服务直接访问，全程在客户端操作。

### Ribbon底层实现原理

Ribbon使用discoveryClient从注册中心读取目标服务信息，对同一接口请求进行计数，使用%取余算法获取目标服务集群索引，返回获取到的目标服务信息。

### SpringCloud有几种调用接口方式

使用Feign和RestTemplate

### DiscoveryClient的作用

可以从注册中心中根据服务别名获取注册的服务器信息。

### 谈谈服务雪崩效应

雪崩效应是在大型互联网项目中，当某个服务发生宕机时，调用这个服务的其他服务也会发生宕机，大型项目的微服务之间的调用是互通的，这样就会将服务的不可用逐步扩大到各个其他服务中，从而使整个项目的服务宕机崩溃.发生雪崩效应的原因有以下几点

1.单个服务的代码存在bug. 2请求访问量激增导致服务发生崩溃(如大型商城的枪红包，秒杀功能). 3.服务器的硬件故障也会导致部分服务不可用.

### 在微服务中，如何保护服务?

一般使用使用Hystrix框架，实现服务隔离来避免出现服务的雪崩效应，从而达到保护服务的效果。当微服务中，高并发的数据库访问量导致服务线程阻塞，使单个服务宕机，服务的不可用会蔓延到其他服务，引起整体服务灾难性后果，使用服务降级能有效为不同的服务分配资源,一旦服务不可用则返回友好提示，不占用其他服务资源，从而避免单个服务崩溃引发整体服务的不可用.

### 服务雪崩效应产生的原因

因为Tomcat默认情况下只有一个线程池来维护客户端发送的所有的请求，这时候某一接口在某一时刻被大量访问就会占据tomcat线程池中的所有线程，其他请求处于等待状态，无法连接到服务接口。

### 谈谈Hystrix服务保护的原理

通过服务降级、服务熔断、服务隔离为高并发服务提供保护。

### 谈谈服务降级、熔断、服务隔离

服务降级：当客户端请求服务器端的时候，防止客户端一直等待，不会处理业务逻辑代码，直接返回一个友好的提示给客户端。

服务熔断是在服务降级的基础上更直接的一种保护方式，当在一个统计时间范围内的请求失败数量达到设定值（requestVolumeThreshold）或当前的请求错误率达到设定的错误率阈值（errorThresholdPercentage）时开启断路，之后的请求直接走fallback方法，在设定时间（sleepWindowInMilliseconds）后尝试恢复。

服务隔离就是Hystrix为隔离的服务开启一个独立的线程池，这样在高并发的情况下不会影响其他服务。服务隔离有线程池和信号量两种实现方式，一般使用线程池方式。

### 服务降级底层是如何实现的？

Hystrix实现服务降级的功能是通过重写HystrixCommand中的getFallback()方法，当Hystrix的run方法或construct执行发生错误时转而执行getFallback()方法。

### 分布式配置中心有那些框架？

Apollo(阿波罗)、zookeeper、springcloud config。

### 分布式配置中心的作用？

动态变更项目配置信息而不必重新部署项目。

### SpringCloud Config 可以实现实时刷新吗？

springcloud config实时刷新采用SpringCloud Bus消息总线。

### 什么是网关?

网关相当于一个网络服务架构的入口，所有网络请求必须通过网关转发到具体的服务。

### 网关的作用是什么

统一管理微服务请求，权限控制、反向代理、负载均衡、路由转发、监控、安全控制黑名单和白名单等

### 网关与过滤器有什么区别

网关是对所有服务的请求进行分析过滤，过滤器是对单个服务而言。

### 常用网关框架有那些？

Nginx、Zuul、Gateway

### Zuul与Nginx有什么区别？

Zuul是java语言实现的，主要为java服务提供网关服务，尤其在微服务架构中可以更加灵活的对网关进行操作。Nginx是使用C语言实现，性能高于Zuul，但是实现自定义操作需要熟悉lua语言，对程序员要求较高，可以使用Nginx做Zuul集群。

### 既然Nginx可以实现网关？为什么还需要使用Zuul框架

Zuul是SpringCloud集成的网关，使用Java语言编写，可以对SpringCloud架构提供更灵活的服务。

### 如何设计一套API接口

考虑到API接口的分类可以将API接口分为开发API接口和内网API接口，内网API接口用于局域网，为内部服务器提供服务。开放API接口用于对外部合作单位提供接口调用，需要遵循Oauth2.0权限认证协议。同时还需要考虑安全性、幂等性等问题。

### ZuulFilter常用有那些方法

Run()：过滤器的具体业务逻辑

shouldFilter()：判断过滤器是否有效

filterOrder()：过滤器执行顺序

filterType()：过滤器拦截位置

### 如何实现动态Zuul网关路由转发

通过path配置拦截请求，通过ServiceId到配置中心获取转发的服务列表，Zuul内部使用Ribbon实现本地负载均衡和转发。

### Zuul网关如何搭建集群

使用Nginx的upstream设置Zuul服务集群，通过location拦截请求并转发到upstream，默认使用轮询机制对Zuul集群发送请求。

## SpringBoot快速开发框架

### 什么是SpringBoot

SpringBoot是快速开发的Spring框架，能够快速整合主流框架，简化xml配置，采用全注解化，内置Http服务器（如tomcat、jetty等），通过java部署运行。

### 为什么要用SpringBoot

快速开发，快速整合，配置简化、内嵌服务容器

### SpringBoot启动方式

主类@SpringBootApplication注解或添加@ComponentScan和@EnableAutoConfiguration注解，使用@SpringBootApplication时注意自动扫描当前包

### SpringBoot与SpringMVC 区别

SpringMVC是SpringBoot的Web开发框架

### SpringBoot与SpringCloud 区别

SpringBoot是快速开发的Spring框架，SpringCloud是完整的微服务框架， SpringCloud依赖于SpringBoot。

### SpringBoot中用那些注解

@EnableAutoConfiguration作用

自动扫描并添加jar包依赖

@SpringBootApplication原理

是一个组合注解，相当于@EnableAutoConfiguration和@ComponentScan

### SpringBoot热部署使用什么？

devtools

### 热部署原理是什么？

热部署的实现原理主要依赖java的类加载机制，在实现方式可以概括为在容器启动的时候起一条后台线程，定时的检测类文件的时间戳变化，如果类的时间戳变掉了，则重新加载整个应用的class文件，同时重启服务，重新部署。

### 热部署原理与热加载区别是什么

热加载是在运行时重新加载class文件，不会重启服务。

### 你们项目中异常是如何处理

在web项目中，使用全局捕获异常返回统一错误信息。

### SpringBoot如何实现异步执行

在启动类添加@EnableAsync表示开启对异步任务的支持，在异步服务上添加@Async

### SpringBoot多数据源拆分的思路

先在properties配置文件中配置两个数据源，创建分包mapper，使用@ConfigurationProperties读取properties中的配置，使用@MapperScan注册到对应的mapper包中

### SpringBoot多数据源事务如何管理

第一种方式是在service层的@TransactionManager中使用transactionManager指定DataSourceConfig中配置的事务

第二种是使用jta-atomikos实现分布式事务管理

### SpringBoot如何实现打包

进入项目目录在控制台输入mvn clean package，clean是清空已存在的项目包，package进行打包

### SpringBoot性能如何优化

如果项目比较大，类比较多，不使用@SpringBootApplication，采用@Compoment指定扫包范围

在项目启动时设置JVM初始内存和最大内存相同

将springboot内置服务器由tomcat设置为undertow

### SpringBoot执行流程

使用SpringApplication.run()启动，在该方法所在类添加@SpringBootApplication注解，该注解由@EnableAutoConfiguration和@ComponentScan等注解组成，@EnableAutoConfiguration自动加载SpringBoot配置和依赖包，默认使用@ComponentScan扫描当前包及子包中的所有类，将有spring注解的类交给spring容器管理

### SpringBoot底层实现原理

使用maven父子包依赖关系加载相关jar包，使用java操作Spring的初始化过程生成class文件，然后用java创建tomcat服务器加载这些class文件

### SpringBoot装配Bean的原理

通过@EnableAutoConfiguration自动获取配置类信息，使用反射实例化为spring类，然后加载到spring容器

## 分布式协调工具

### 什么是ZooKeeper

ZooKeeper是Java语言编写的开源框架，用以协调分布式的一个工具。

### ZooKeeper存储结构与特性

类似于树形结构，同一层节点名称不能重复。节点类型分为临时节点与持久节点

Zookeeper以节点方式进行存储，类似于xml树状结构；

1. 节点又分为节点名称（全路径不能重复）和 节点值

节点类型有持久节点（持久化在硬盘上）和临时节点（会话与临时节点同死同生）

b、节点功能：每个节点都有通知功能，当这个节点增删改的时候都会有事件通知

Zookeeper主要有一下特性

a、一致性：数据按照顺序分批入库；

b、原子性：事务要么成功要么失败，不会局部化；

c、单一视图：客户端连接集群中任何一个zk节点，数据都是一致的；

d、可靠性：每次对zk的操作状态都会保存在服务端；

e、实时性：客户端可以读取到zk服务器的 新数据；

### ZooKeeper中临时节点与持久节点区别

持久节点是持久化在硬盘上，会话断开后节点也能查到；

临时节点与会话保持连接，会话在节点在，会话断开，节点也会删除；

### ZooKeeper应用场景

1. 服务注册与发现的中心

B、利用临时节点特性解决分布式锁

C、分布式配置中心

D、基于哨兵机制实现选举策略

E、实现本地负载均衡

F、基于节点事件通知特性可做消息中间件

G、分布式事务

### 什么场景下会导致ZooKeeper发生延迟通知

watch事件延迟：节点被修改后，会有事件通知发往观察者，直到接收到watch事件，观察者才会知道节点被修改了；当管擦着接到watch事件的那一刻，该节点又被其他修改者修改了，而 近的watch事件还没有通知到观察者，就会造成延迟通知。

### 分布式锁有那些实现方案

1. 基于setNx实现分布式锁（麻烦，需要考虑死锁及释放问题）

b、redission实现分布式锁

c、zookeeper实现分布式锁（基于临时节点，实现简单，效率高，失效时间容易控制）

### ZooKeeper实现分布式锁的原理

多个jvm在同一个zookeeper上创建同一个节点（临时节点），哪个jvm能创建成功，就表示它拿到了锁，剩下的jvm保持对这个节点的监听，一旦发现这个节点被删除了，那么剩下的jvm就重新再创建这个节点，谁能创建成功谁能拿到锁，依次循环下去。

### ZooKeeper实现分布式锁与Redis实现分布式锁区别

Zookeeper通过创建临时节点和利用监听事件实现分布式锁，Redis使用setnx命令创建相同的key，因为Redis的key保证唯一，先创建的先获取锁。

不断的去尝试，去获取锁，比较耗性能

Zookeeper实现分布式锁，即使获取不到锁，创建对锁的监听即可，不需要不断去尝试获取 锁，性能开销小

Redis实现分布式锁，如果客户端获取到锁的时候遇到bug或挂了，还需要等到超时时间过了以后才能重新获取锁

Zookeeper实现分布式锁，创建的是临时节点，客户端挂了，节点自然删除，也就达到了自动释放锁的效果

### 使用Zookeeper实现服务Master选举原理

多个服务器在启动时候，会在Zookeeper上创建相同的临时节点，谁如果能够创建成功，谁就为主。如果主服务器宕机，其他备用节点获取监听信息，重新创建节点，选出主服务器。

### ZooKeeper集群选举原理

每台Zookeeper服务器启动时会发起投票，每次投票后，服务器统计投票信息，如果有机器获取半数以上的投票数则leader产生。

## 微服务常用解决方案

### 谈谈网站跨域解决方案

a、使用jsonp 缺点只能发送get请求

b、使用httpclient进行转发，效率低

c、设置响应头允许跨域

d、使用Nginx搭建api网关

e、使用Zuul微服务搭建api接口网关

### 分布式Session一致性问题

a、使用Nginx反向代理，即IP绑定，同一个ip只能在同一个机器上访问

b、使用数据库，但性能不高

c、tomcat内置了对session同步的支持，但可能会产生延迟

d、使用Spring-Session框架，相当于把session放到redis中

e、使用token令牌代替session

### 分布式锁解决方案

### 分布式事务解决方案

### 分布式任务调度平台解决方案

### 分布式事务解决方案

### 分布式日志收集解决方案

### 分布式全局id生成方案

### 分布式服务追踪与调用链系统解

## 分布式消息系统与中间件

### 消息中间件产生的背景

传统的Web项目采用http协议基于请求和响应传输信息，请求发出后必须等待服务器端响应，如果服务器端不会及时响应客户端会一直等待。

### Http协议同步接口调用失败了怎么做？

采用消息补偿机制重新发送请求。

### 消息队列异步通讯与同步通讯区别

同步通讯是客户端直接将请求发往服务器，等待服务器处理完请求并返回响应信息后才会继续向下执行。消息队列独立于客户端和服务器端，单独架设消息队列服务器，对于不必立即获取响应和处理过程复杂的请求，客户端可以将请求发往消息队列然后立即返回，指定的消费者处理请求，这样客户端不必持续等待。

### JMS消息通讯模型有那些

消息队列：点对点，发布订阅

### 消息中间应用场景

发送邮件或短信的服务、秒杀、运行过程复杂耗时的服务。

### 发布订阅与点对点通讯的区别

发布订阅是生产者发布一个主题，所有订阅该主题的消费者都会参与消费，消息会被重复消费。点对点通讯是一个消息只能有一个消费者消费，需要保证数据的幂等性。

### 如何保证JMS可靠消息

ActiveMQ采用消息签收机制保证数据的可靠性，消息签收有三种方式：自动签收、手动签收、事务，默认自动签收。如果是带事务的消息，事务执行完毕后自动签收，事务回滚则重新发送。

### ActiveMQ服务端宕机了，消息会丢失吗？

生产者可以通过setDeliveryMode方法设置消息模式，当设置未非持久化时服务器宕机后消息将销毁，重启服务器后无法继续消费。当设置为持久化时服务器宕机后消息将保存到服务器中，重启后消费者还可以继续消费未处理完毕的消息。

### RabbitMQ与其他MQ有什么不同？

RabbitMQ是用erlang语言实现，安装RabbitMQ之前需要先安装erlang环境。RabbitMQ只支持AMQP协议，用于对稳定性要求比较高的企业。

### RabbitMQvirtualHost的作用

VirtualHost相当于是虚拟的RabbitMQ服务器，每个VirtualHost都是独立的，起到隔离交换机、队列的作用，不同的项目组可以连接到不同的VirtualHost不会互相影响。

### 谈谈RabbitMQ五种列队形式

点对点模式：生产者生成的消息由一个消费者消费；

工作模式：在消费者集群的情况下，可以根据消费者服务器的性能分配消息，即性能好的服务器多消费，性能次的少消费。

发布订阅模式：在生产者和队列之间插入一个交换机，由交换机转发到与该交换机绑定的队列；

路由模式：路由模式是基于发布订阅模式，只是在生产者向交换机生产消息时板顶一个routingkey，交换机向绑定同一个routingkey的队列转发消息；

通配符模式：是对路由模式的补充，使用通配符进行routingkey匹配，通配符有#和\*，#代表匹配多个，\*代表匹配一个。

### RabbitMQ四中交换机类型

Fanout：以这种方式连接到交换机的队列都可以获得交换机的转发；

Direct：生产者绑定direct类型的交换机，在向交换机发送消息时绑定routingkey，交换机会将这条消息发送到相同routingkey的队列。

Topic：和Direct相似，可以在routingKey中使用通配符，#代表多个匹配，\*代表单个匹配，routingkey使用”.”作为分隔符。

Headers：类似Direct，使用多个消息代替路由键建立路由规则，通过消息头匹配来转发消息。

### RabbitMQQMAP协议原理

在RabbitMQ中，生产者将消息发送到交换机，交换机将消息根据路由策略将消息发送到绑定的消息队列，消费者通过消息队列获取并消费消息。

# 高并发解决方案

## 高性能Nginx+Lua

### 什么是DNS解析域名

DNS域名解析就是讲域名转化为不需要显示端口（二级域名的端口一般为80）的IP地址，域名解析的一般先去本地环境的host文件读取配置，解析成对应的IP地址，根据IP地址访问对应的服务器。若host文件未配置，则会去网络运营商获取对应的IP地址和域名.

### 你用过那些外网映射工具

花生壳,natapp,ngrok。

### 什么是Nginx

Nginx是一个高级的轻量级的web服务器,由俄罗斯科学家开发的，具有如下优点:

1.占用内存少，并发量强，支持多种并发连接，效率高.

2.能够作为负载均衡服务器和(内部直接支持 Rails 和 PHP)代理服务器。Nginx用C编写开销和CPU占有小.

3.安装启动简单，配置简洁，bug少，一般几个月不需要重新启动且不会宕机，稳定性和安全性好.

### Nginx的作用

反向代理、负载均衡、配置主备tomcat、动静分离

### Nginx 应用场景

做HTTP服务器、反向代理服务器、静态资源服务器

### 什么是反向代理

代替真实服务器接收网络请求，然后将请求转发到真实服务器

### 反向代理的作用

隐藏真实服务器，使真实服务器只能通过内网访问，保护了真实服务器不被攻击。配置负载均衡，减轻单台真实服务器的压力。配置主备服务器，保持服务稳定运行。

### Nginx如何配置反向代理

首先到DNS服务器做域名解析，如果是局域网在hosts文件中配置IP和域名对应关系。编辑nginx的nginx.conf文件，配置server\_name为指向nginx服务器的域名，location拦截请求，如果是访问nginx本地资源则配置root，如果是反向代理到真实服务器则配置proxy\_pass为服务器地址

### 说说常用Nginx的相关配置

upstream 负载均衡配置

server [IP] [weight] [backup] 配置tomcat集群

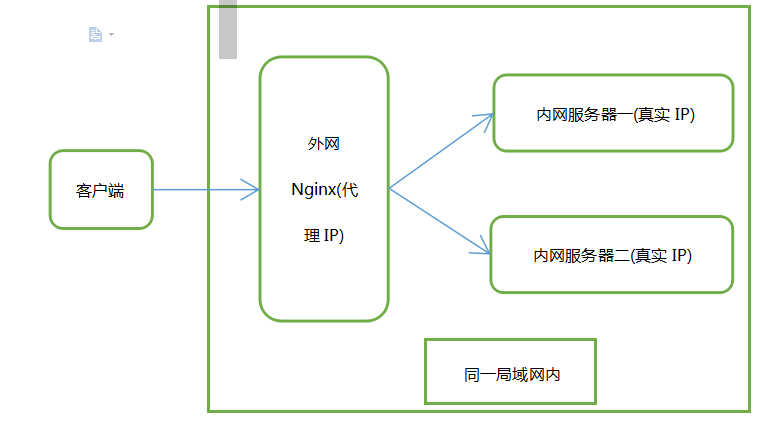
proxy\_connect\_timeout、proxy\_read\_timeout、proxy\_send\_timeout 连接时间、真实服务器响应时间、返回结果时间

location 匹配用户请求的url

root 配置本地资源路径

proxy\_pass 配置真实服务器地址

### 请画图展示反向代理流程



### LVS与Nginx区别

LVS是四层反向代理，基于TCP和UDP协议，可用于管理Nginx集群，抗负载能力强。Nginx是七层反向代理，基于HTTP协议，用于管理真实服务器集群。

### location的作用

匹配用户请求url，根据不同请求转发到不同的服务器。

### Nginx中如何配置负载均衡

在upstream中配置多个server，在location的proxy\_pass配置为http://+upstream名称

### 四层负载均衡与七层负载均衡区别

四层负载均衡基于TCP和UDP协议，通过IP+端口号接受请求并转发到服务器。七层负载均衡基于HTTP协议，通过url或主机名接收请求并转发到服务器。

### 四层负载均衡有那些实现方案

LVS、F5

### 负载均衡有那些算法

轮询算法:按照时间顺序分配到不同的服务器，当其中一台服务器宕机则被自动剔除，切换到正常的服务器。

权重算法:按照分配给服务器的权重比例来分发到不同服务器，权重比例越高，则访问几率越大。

IP绑定(ip\_hash):根据访问的IP的哈希结果来判定，使同一个IP访问一台固定的后端服务器，同时解决动态页面的session问题.

### 服务器集群后，会产生了那些问题

分布式锁

分布式全局ID

分布式Session一致性问题

分布式事务

分布式任务调度

分布式日志收集

分布式配置中心

### 什么是动态负载均衡

一般情况下，使用nginx搭建服务器集群，每次修改nginx.conf配置文件都需要重启nginx服务器。动态负载均衡就是修改nginx.conf配置文件后不必重启nginx而使配置生效。

### Nginx如何实现动态负载均衡

搭建Nginx+Consul+Upsycn环境。Nginx实现服务的反向代理和负载均衡。Consul是一个开源的注册中心和服务发现的框架，通过HTTP API来发现服务，注册服务。同时支持故障发现，K/V存储，多数据中心，Raft算法等多中高可用特性。Consul在Nginx动态负载均衡作用是

通过Http api注册和发现服务.Upsycn是新浪微博的开源框架，在Nginx动态负载均衡的作用是Consul的后端的server列表，即获取Nginx的上游服务器(Upstream server）信息,并动态更新Nginx的路由信息.

### 什么是Http协议

超文本传输协议

### Http协议组成部分

Http协议是基于TCP协议封装成超文本传输协议，包括请求(request）和响应(response),http协议请求（request）分为请求参数（request params）和方法类型(request method）、请求头（request hearder）、请求体(request body) ，

响应(response)分为 响应状态(response state）、响应头（response header）、响应体(response body）等.

### TCP与UDP区别

udp:

a、是面向无连接, 将数据及源的封装成数据包中,不需要建立连接

    b、每个数据报的大小在限制64k内

    c、因无连接,是不可靠协议

    d、不需要建立连接,速度快

tcp：

a、建议连接，形成传输数据的通道.

    b、在连接中进行大数据量传输，以字节流方式

    c 通过三次握手完成连接,是可靠协议

    d 必须建立连接m效率会稍低

### 谈谈七层网络模型

应用层：客户端的各种应用、app；

表示层：进行数据的格式区分，如图片、编码；

会话层：本地主机与远程主机的会话管理；

传输层：定义传输数据的协议端口号，TCP和UDP是这一层的协议；

网络层：进行逻辑地址寻址；

数据链路层：建立逻辑连接，进行硬件地址寻址；

物理层：建立物理连接；

### Nginx如何实现TCP四层负载均衡

在nginx.conf文件中配置tcp模块，在upstream块中定义socket服务器负载均衡，其余与nginx配置七层负载均衡相同。

tcp {

### 定义多个上游服务器

upstream itmayeidu{

### 定义TCP模块上游服务器

server 192.168.5.165:80001;

server 192.168.5.165:80002;

}

server {

listen 9999;

server\_name 192.168.212.137;

### 反向代理upstream

proxy\_pass itmayeidu;

}

}

### lvs 与Nginx 区别

lvs工作在网络第四层，nginx工作在网络第七层；lvs比nginx抗负载能力强；lvs对网络依赖性强，nginx对网络依赖性弱；lvs几乎可以对所有应用做负载均衡，比如数据库。

### lvs与keepalived区别

Lvs可以实现负载均衡，但是无法实现健康检查。Keepalived可以进行健康检查实现高可用。

### keepalived 作用

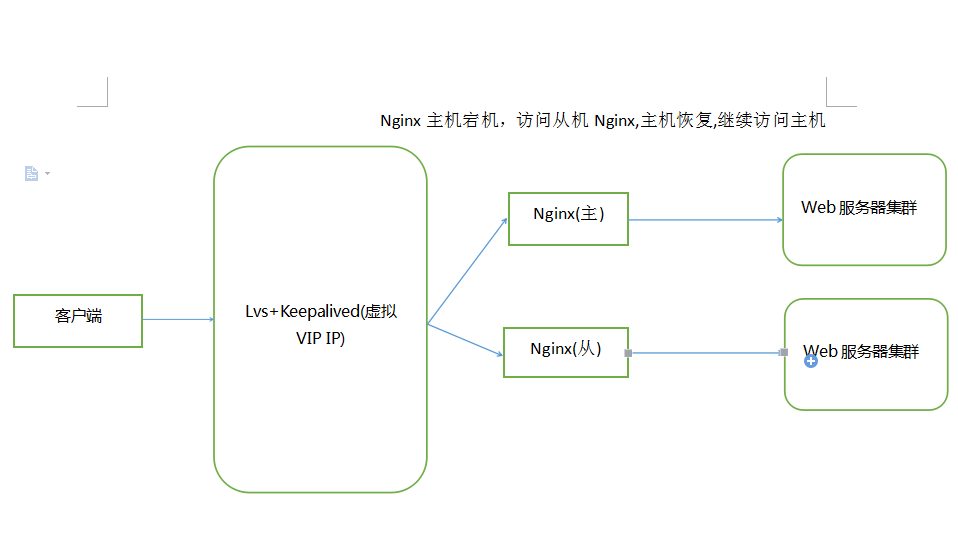
keepalive 软件可以进行健康检查，而且能同时实现 LVS 的高可用性，解决 LVS 单点故障的问题

### 如何实现双机主从热备

Nginx+Tomcat：在upstream中配置多台服务器，从服务器后加backup

Keepalived+Nginx：在多台nginx服务器上安装keepalived，将主服务器的state设置为MASTER，从服务器设置为BACKUP，主服务器的优先级要高于从服务器

### lvs+Keepalived+Nginx架构流程图



### 项目发布如何不影响到正常用户访问，实现7\*24小时访问

可以两台机子互为热备，平时各自负责各自的服务。在做上线更新的时候，关闭一台服务器的tomcat后，nginx自动把流量切换到另外一台服务的后备机子上，从而实现无痛更新，保持服务的持续性，提高服务的可靠性，从而保证服务器7\*24小时运行。

### 项目如何发生故障宕机了，如何处理。

使用lvs+keepalived+Nginx做主从热备，lvs管理nginx集群，nginx管理服务器集群，在服务器宕机的情况下keepalived启动健康检测，多次重启无果可以短信通知运维人员及时维护。

### 动态网站与静态网站区别

在浏览器中打开一个网站，点击鼠标右键查看源码，多次请求后如果源码不产生变化就是静态网站，变化就是动态网站。

### 动态页面静态化的作用

便于搜索引擎抓取和排名

### 什么是动静分离架构模式

静态页面与动态页面分开不同系统访问的架构设计方法，静态页面与动态页面以不同域名区分。

### 如何搭建动静分离

以nginx服务器作为静态资源服务器，静态资源和动态资源访问分开配置，静态资源在location中使用本地文件路径配置方式，动态资源使用proxy\_pass配置到后台服务器。

如：

###静态资源访问

server {

listen 80;

server\_name static.itmayiedu.com;

location /static/imgs {

root F:/;

index index.html index.htm;

}

}

###动态资源访问

server {

listen 80;

server\_name www.itmayiedu.com;

location / {

proxy\_pass http://127.0.0.1:8080;

index index.html index.htm;

}

}

### 动静分离与前后分离区别

动静分离是将静态资源和动态资源存放在不同服务器中，前后分离是将前端和后台分离，前端通过api调用后台接口

### 如何控制浏览器静态资源缓存

静态资源存在缓存的原因是项目上线时，浏览器缓存中的静态资源导致与服务器将淘汰资源的代码发生冲突(或者是页面访问频繁访问同一资源，导致一些浏览器如IE(本人开发亲身经历过)返回默认的响应结果，与实际响应结果不符合)，

一般的服务器是强制F5进行刷新或者是清除缓存，最有效的解决方法就是在请求资源后面加上变量(如时间戳,随机数)

### Http状态码304的作用

表示浏览器存在静态资源缓存就不从服务器获取静态资源

## 高并发服务限流特级

### 高并发服务限流特技有哪些算法?

传统计算器算法，滑动窗口计数器算法,令牌桶算法和漏桶算法。

### 传统计数器限流算法有什么弊端?

传统计数器限流方式不支持高并发，存在线程安全问题.若大量访问请求集中在计数器最后时刻，计数器极易发生临界问题，访问的请求无法完成.

### 什么是滑动窗口计数器?

滑动窗口计数器是一种服务限流的算法,相对于计数器方法的实现，滑动窗口实现会更加平滑，并自动消除毛刺。其原理是当有访问进来时，会判断若干个单位来的请求是否超过

设置的阀值，并对当前时间片的请求数+1.

### 令牌桶算法的原理?

向一个存放固定容量令牌的同，以固定速率往桶里添加令牌，当桶已经装满时，新增的令牌会被丢弃或者拒绝,当一个固定数目的数据包到达时，会在

桶中删除同等数量的令牌，数据包会发到网络上，当这个固定数目超过桶中的令牌数，不会删除桶中的令牌数目，则该数据包会被限流(丢弃或者存入缓冲区等待)

### 漏桶算法的原理?

向一个存放固定容量的桶，以任意速率滴入水滴(请求)，以固定速率滴出水滴，当滴入水滴量超过桶中设置固定容量，则会发生溢出，溢出的水滴的请求是无法访问的，直接走

服务限流降级，桶中的容量不发生任何变化。

### 令牌桶与漏桶算法的区别?

令牌桶和漏桶算法的区别是令牌桶会根据请求的令牌数与桶中的令牌数做对比，倘若桶中令牌数小于请求令牌数则多余的令牌数的请求被拒绝。漏桶算法则是向桶中添加请求，当

请求数大于桶中容量发生溢出，溢出的请求直接被拒绝访问。主要区别是漏桶算法是强行限制数据的传输速率，而令牌桶在能够限制数据的平均传输速率外，还允许某种程度的突发传输，使用于抢红包等高并发的场景。

## Web前端优化方案

### Web前端有哪些优化方案

1.网站框架实行动静分离，动态资源和和静态资源部署到不同的服务器上面，使用nginx访问本地静态资源，通过nginx代理转发到tomcate访问动态资源

2.在访问静态资源时在Url后缀加上时间戳，防止访问资源的与浏览器本地缓存资源存在冲突.

3.页面减少HTTP请求，合并静态资源(如js或者css）并进行压缩。

4.使用CDN内容分发，缓存静态资源，让用户访问最近的服务器，减少宽带之间的传输.

### 什么是CDN内容分发?

CDN内容转发是指在用户和服务器之间加上一个缓存机制，一组分布在不同的静态资源服务器，储存静态资源，动态获取用户IP，并让用户访问最近的静态资源服务器,防止出现网络延迟阻塞，

提高访问速度。CND服务器部署在网络运营商的机房，用户请求首先会访问CND服务器，如果CND服务器中没有缓存会自动把创建缓存，当用户再次访问时，直接获取缓存资源，并返回给前端，提高静态

资源的访问速度。

### CDN内容加速原理?

1)用户向浏览器提供要访问的域名.

2)浏览器从本地host文件中解析域名,如何host文件没有做任何配置，则浏览器调用域名解析库对域名进行解析,析函数库一般得到的是该域名对应的CNAME记录,从中获取真正的IP地址,此过程中,根据地理位置信息解析对应的IP地址，使得用户能就近访问静态资源服务器。

3)此次解析的CDN服务器的IP地址，浏览器根据这个地址，向缓存服务器发出请求。

4)缓存服务器根究浏览器访问的域名，通过缓存内部获得此域名的真实IP地址，再有缓存服务器提交该IP地址的访问请求。

5)缓存服务器从服务器的ip地址获取内容备用,并将数据返回给用户，完成服务过程.

6)客户端将服务器返回的数据展示给前端

## 互联网高并发解决方案

首先在谈到高并发解决方案的时候，很多学员可能都会想到服务器应该做集群、负载均衡。

那么服务器集群，一定能解决高并发吗？这其实不一定。

首先要分清楚高并发影响用户的源头？是因为带宽不够还是服务内存不足？

服务带宽指的是：客户端与服务器传输的宽度的速度，1m 等于 128kb。

服务内存指的是：服务器端处理业务能力。

那么解决高并发的入口是客户端与服务器端传输带宽速度, 如果带宽速度不足的情况，可能会导致客户端延迟等待。

一个网站核心 分为静态资源(css、img、js)和动态资源（jsp、ftl）组合，绝大数的情况下静态资源占了整个网站带宽传输， 这时候应该采用网站动静分离架构，将动态资源与静态资源分开服务器存放，注意：网站跨域问题。

动静分离可以使用 nginx，或者是使用第三方静态资源服务器比如七牛云、阿里云等。

还要对静态资源实现压缩、减少带宽的传输，使用 CDN 实现内容分发，从最近服务器访问。

如何对静态资源实现压缩呢？ 使用 nginx 开启 gzip、maven 打包压缩 min 格式、或者使用 cdn 自带压缩

以上这些属于 Web 前端优化。

如果客户端发送请求已经达到服务器端的话，服务端处理响应产生延迟，那么开始采用后端优化方案。

.可以对服务器实现集群 、加服务配置、采用 MQ 异步传输、使用 Redis 做缓存，减轻数据库访问压力、代码优化、数据库采用:读写分离和分表分库，程序采用多线程、jvm 参数调优，服务实现保护机制（服务降级、服务隔离、服务熔断、服务限流）等。

Web 前端优化大多数情况下，属于公司运维干的事情，后端优化属于架构师做的事情，如果一个网站中静态资源非常多的情况下，不要将静态资源和动态资源在同一个服务器存放，一定要采用动静分离架构，提高网站的吞吐量。

最后总结下，如果服务器带宽不足的情况下，服务器接受客户端请求资源，可能会产生延迟，服务器做集群、加配置，效果不会很明显，因为服务器集群只能提高服务器的业务处理能力，不能提高服务器的带宽传输，

所以可以采用以上总结的 Web 前端优化方案，减少客户端与服务器端带宽传输。

.如果在带宽的足够的情况下，客户端发送请求已经到达了后端服务器，服务器端处理能力产生延迟，那么采用以上总结 后端优化方案

服务器集群、加服务器配置等。

之前有一位学员问，app 客户端遇到高并发，是采用后端优化还是前端优化，app 属于 cs 架构，静态资源在打包的时候已经在安装包里面，不需要远程获取，业务逻辑需要远程调用接口，获取 json 数据进行解析，让后展示数据，所以 app 客户端产生的高并发，核心在于后端优化。