**高并发与高可用实战**

# 补充基础知识

### DNS域名解析

整个过程大体描述如下，其中前两个步骤是在本机完成的，后8个步骤涉及到真正的域名解析服务器：1、浏览器会检查缓存中有没有这个域名对应的解析过的IP地址，如果缓存中有，这个解析过程就结束。浏览器缓存域名也是有限制的，不仅浏览器缓存大小有限制，而且缓存的时间也有限制，通常情况下为几分钟到几小时不等，域名被缓存的时间限制可以通过TTL属性来设置。这个缓存时间太长和太短都不太好，如果时间太长，一旦域名被解析到的IP有变化，会导致被客户端缓存的域名无法解析到变化后的IP地址，以致该域名不能正常解析，这段时间内有一部分用户无法访问网站。如果设置时间太短，会导致用户每次访问网站都要重新解析一次域名。

2、如果用户浏览器缓存中没有数据，浏览器会查找操作系统缓存中是否有这个域名对应的DNS解析结果。其实操作系统也有一个域名解析的过程，在Windows中可以通过C:\Windows\System32\drivers\etc\hosts文件来设置，在Linux中可以通过/etc/hosts文件来设置，用户可以将任何域名解析到任何能够访问的IP地址。例如，我们在测试时可以将一个域名解析到一台测试服务器上，这样不用修改任何代码就能测试到单独服务器上的代码的业务逻辑是否正确。正是因为有这种本地DNS解析的规程，所以有黑客就可能通过修改用户的域名来把特定的域名解析到他指定的IP地址上，导致这些域名被劫持。

3、前两个过程无法解析时，就要用到我们网络配置中的"DNS服务器地址"了。操作系统会把这个域名发送给这个LDNS，也就是本地区的域名服务器。这个DNS通常都提供给用户本地互联网接入的一个DNS解析服务，例如用户是在学校接入互联网，那么用户的DNS服务器肯定在学校；如果用户是在小区接入互联网，那么用户的DNS就是再提供接入互联网的应用提供商，即电信或联通，也就是通常说的SPA，那么这个DNS通常也会在用户所在城市的某个角落，不会很远。**Windows环境下通过命令行输入ipconfig，Linux环境下通过cat /etc/resolv.conf**就可以查询配置的DNS服务器了。这个专门的域名解析服务器性能都会很好，它们一般都会缓存域名解析结果，当然缓存时间是受到域名的失效时间控制的。大约80%的域名解析到这里就结束了，所以LDNS主要承担了域名的解析工作。

4、如果LDNS仍然没有命中，就直接到Root Server域名服务器请求解析

5、根域名服务器返回给本地域名服务器一个所查询的主域名服务器（gTLD Server）地址。gTLD是国际顶级域名服务器，如.com、.cn、.org等，全球只有13台左右

6、本地域名服务器LDNS再向上一步返回的gTLD服务器发送请求

7、接受请求的gTLD服务器查找并返回此域名对应的Name Server域名服务器的地址，这个Name Server通常就是用户注册的域名服务器，例如用户在某个域名服务提供商申请的域名，那么这个域名解析任务就由这个域名提供商的服务器来完成

8、Name Server域名服务器会查询存储的域名和IP的映射关系表，在正常情况下都根据域名得到目标IP地址，连同一个TTL值返回给DNS Server域名服务器

9、返回该域名对应的IP和TTL值，LDNS会缓存这个域名和IP的对应关系，缓存时间由TTL值控制

10、把解析的结果返回给用户，用户根据TTL值缓存在本地系统缓存中，域名解析过程结束

在实际的DNS解析过程中，可能还不止这10步，如Name Server可能有很多级，或者有一个GTM来负载均衡控制，这都有可能会影响域名解析过程。

# 高并发与高可用

## 大型网站系统应有的特点

### 高并发，大流量

高并发，大流量：需要面对高并发用户，大流量访问。举个例子，去往迪拜的飞机有200张票，但是有100w人都挤进系统买票，如何让这100w人能够看到票务的实时更新，以及顺利的买到一张票，都是一个网站架构师应该考虑的问题。这也许对于淘宝的“双十一”1000w的一分钟独立访问用户量来说，是个微不足道的数字，但是对于用户的体验以及网站的口碑来说，都是一项不小的挑战。

### 高可用

高可用：相对于高并发来说，高可用并不是一个比较有规律的参数，7\*24 是每个网站的梦想，但是你并不知道，在某一刻，他就没理由的宕机了。

### 海量数据

海量数据：存储，管理海量的数据，需要使用大量的服务器。FaceBook每周上传的照片接近10亿，没有一个大型的存储服务器的支撑，相信用户量不会一直飙升。

### 用户分布广泛，网络情况复杂

用户分布广泛，网络情况复杂：许多大型的互联网都是为全球用户提供服务的，用户分布范围广，各地网络情况千差万别。各个运行商之间的互通，各个国家的数据连接等等。

### 安全环境恶劣

安全环境恶劣：由于互联网的开放性，使得互联网更容易受到攻击，包括各种省份证信息被窃取等事件屡见不鲜。

### 渐进式发展

渐进式发展：几乎所有的大型互联网网站都是从一个小网站开始，渐进发展起来的，好的互联网产品都是慢慢运营出来的。

## 网站架构演变过程

### 传统架构

传统项目分为三层架构，将业务逻辑层、数据库访问层、控制层放入在一个项目中 使用SSH或者SSM技术。

优点:适合于个人或者小团队开发，不适合大团队开发。

### 分布式架构

根据业务需求进行拆分成N个子系统，多个子系统相互协作才能完成业务流程子系统之间通讯使用RPC远程通讯技术。

优点:

1.把模块拆分，使用接口通信，降低模块之间的耦合度。

2.把项目拆分成若干个子项目，不同的团队负责不同的子项目。

3.增加功能时只需要再增加一个子项目，调用其它系统的接口就可以。

4.可以灵活的进行分布式部署。

有优点就有缺点，缺点如下：

1.系统之间交互需要使用远程通信，接口开发增加工作量。

2.各个模块有一些通用的业务逻辑无法共用。

为了解决上面分布式架构的缺点，我们引入了soa架构，SOA：Service Oriented Architecture面向服务的架构。也就是把工程拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。

### SOA架构

SOA是一种软件架构模式，将共同的业务逻辑抽取出来，封装成单独的服务

业务系统分解为多个组件，让每个组件都独立提供离散，自治，可复用的服务能力

通过服务的组合和编排来实现上层的业务流程

作用：简化维护,降低整体风险,伸缩灵活

### 微服务架构

微服务是指开发一个单个、小型的但有业务的服务，每个服务都有自己的处理和轻通讯机制，可以部署在单个服务器上，让专业的人做专业的事情。

微服务与SOA相比，更加轻量级。

### SOA与微服务架构区别

OA架构主要针对企业级、采用ESB服务（ESB企业服务总线），非常重，需要序列化和反序列化，采用XML格式传输。

微服务架构主要互联网公司，轻量级、小巧，独立运行，基于Http+Rest+JSON格式传输。

ESB也可以说是传统中间件技术与XML、Web服务等技术相互结合的产物。

1.在微服务中，与SOA不同，服务可以独立于其他服务进行操作和部署，因此更容易经常部署新版本的服务和独立扩张服务，让专业的人做专业的事情，快速迭代新的产品。

2.在SOA中服务可能共享数据存储，而微服务中每个服务都具有独立的数据存储。

3.SOA与微服务主要区别在于规模和范围，SOA是一种思想，是面向服务架构体系，微服务继承 了SOA的优点，去除传统的ESB消息总线，采用Http+json格式通讯方式，更加轻量级。

## 高并发设计原则

系统设计不仅需要考虑实现业务功能，还要保证系统高并发、高可用、高可靠等。同时还应考虑系统容量规划（流量、容量等）、SLA指定（吞吐量、响应时间、可用性、降级方案等）、监控报警（机器负载、响应时间、可用率等）、应急预案（容灾、降级、限流、隔离、切流量、可回滚等）。

缓存

异步并发

连接池

线程池

扩容

消息队列

分布式任务

### 拆分系统

在我们从零开始做一个新系统的时候，会首先进行系统功能模块架构设计，那么是直接做一个大而全的垂直的MVC系统，使用一个war包进行发布管理，还是需要按一些规则进行模块拆分，设计成SOA或者微服务系统比较好呢？这个笔者认为需要依据项目具有什么样的人力物力条件以及项目需要支撑多少用户量和交易量为基础。一个好的系统设计应该能够满足解决当前的需求和问题，把控实现和进度风险，预测和规划未来，避免过度设计，在上线一个基础核心版本之后，再进行不断迭代和完善。

今天我们来谈一谈进行SOA、微服务系统架构设计时模块拆分的一些维度和原则。

系统维度：按照系统功能、业务拆分，如、优惠券、购物车，结算，订单等系统。

功能维度：对系统功能在做细粒度拆分，优惠券系统分为 优惠券后台系统、领券系统、发券系统。

读写维度：比如商品系统中，如果查询量比较大，可以单独分为两个服务，分别为查询服务和写服务,

读写比例特征拆分；读多，可考虑多级缓存；写多，可考虑分库分表.

AOP 维度： 根据访问特征，按照 AOP 进行拆分，比如商品详情页可分为 CDN、页面渲染系统，CDN 就是一个 AOP 系统

模块维度：对整体代码结构划分 Web、Service、DAO

### 服务化

在分布式系统中，将业务逻辑层封装成接口形式，暴露给其他系统调用，那么这个接口我们可以理解为叫做服务。

当服务越来越多的时候，就会需要用到服务治理，那么会用到Dubbo、SpringCloud服务治理框架

后续在深入Dubbo和SpringCloud中会详细讲到。

服务化演进: 进程内服务-单机远程服务-集群手动注册服务-自动注册和发现服务-服务的分组、隔离、路由-服务治理

考虑服务分组、隔离、限流、黑白名单、超时、重试机制、路由、故障补偿等

实践：利用 Nginx、HaProxy、LVS 等实现负载均衡，ZooKeeper、Consul 等实现自动注册和发现服务

### 消息队列

消息中间件是一个客户端与服务器异步通讯框架，消息中间件中分为点对点与发布订阅通讯方式，生产者发送消息后，消费者可以无需等待， 异步接受生产者发送消息。

在电商系统中，会使用消息队列异步推送消息，注意消息失败重试幂等性问题。

幂等性问题解决方案，使用持久化日志+全局id记录。

### 缓存技术

浏览器端缓存

APP客户端缓存

CDN(Content Delivery Network)缓存

接入层缓存

应用层缓存

分布式缓存

对于兜底数据或者异常数据，不应该让其缓存，否则用户会在很长一段时间里看到这些数据。

### 并发化

改串行为并行。

## 高可用设计原则

通过负载均衡和反向代理实现分流。

通过限流保护服务免受雪崩之灾。

通过降级实现部分可用、有损服务。

通过隔离实现故障隔离。

通过合理设置的超时与重试机制避免请求堆积造成雪崩。

通过回滚机制快速修复错误版本。

### 降级

对于高可用服务，很重要的一个设计就是降级开关，在设计降级开关时，主要依据如下思路：

1.开关集中化管理：通过推送机制把开关推送到各个应用。

2.可降级的多级读服务：比如服务调用降级为只读本地缓存、只读分布式缓存、只读默认降级数据（如库存状态默认有货）。

3.开关前置化：如架构是Nginx–>tomcat，可以将开关前置到Nginx接入层，在Nginx层做开关，请求流量回源后端应用或者只是一小部分流量回源。

4.业务降级：当高并发流量来袭，在电商系统大促设计时保障用户能下单、能支付是核心要求，并保障数据最终一致性即可。这样就可以把一些同步调用改成异步调用，优先处理高优先级数据或特殊特征的数据，合理分配进入系统的流量，以保障系统可用。

### 限流

目的: 防止恶意请求攻击或超出系统峰值

实践：

恶意请求流量只访问到 Cache

穿透后端应用的流量使用 Nginx 的 limit 处理

恶意 IP 使用 Nginx Deny 策略或者 iptables 拒绝

### 切流量

目的：屏蔽故障机器

实践:

DNS: 更改域名解析入口，如 DNSPOD 可以添加备用 IP，正常 IP 故障时，会自主切换到备用地址; 生效实践较慢

HttpDNS: 为了绕过运营商 LocalDNS 实现的精准流量调度

LVS/HaProxy/Nginx: 摘除故障节点

### 可回滚

发布版本失败时可随时快速回退到上一个稳定版本

## 业务设计原则

### 防重设计

页面请求防止重复提交，可以采用防重key、放重表、Token等

采用图形验证，防止机器攻击。

### 幂等设计

消息中间件:消息中间件中应该注意因网络延迟的原因，导致消息重复消费

第三方支付接口:在回调接口中，应该注意网络延迟，没有及时返回给第三方支付平台，注意回调幂等性问题。

分布式系统中，保证生成的订单号唯一性，定时Job执行的幂等性问题等。

### 流程定义

复用流程系统，提供个性化的流程服务。

### 状态与状态机

复用流程系统，提供个性化的流程服务。

### 后台系统操作可反馈

设计后台系统时，考虑效果的可预览、可反馈。

### 后台系统审批化

对于有些重要的后台功能需要设计审批流，比如调整价格，并对操作进行日志记录，从而保证操作可追溯、可审计。

### 文档注释

系统发展的最初阶段就应该有文档库（设计架构、设计思想、数据字典/业务流程、现有问题），业务代码合特殊需求都要有注释。

### 备份

包括代码和人员的备份。代码主要提交到代码仓库进行管理和备份，代码仓库至少应该具备多版本的功能。人员备份指的是一个系统至少应该有两名开发人员是了解的。

# 环境准备

CentOS7 7.0 64位 以上+一台外网服务器+一个域名+CDN内容分发

电脑配置 16g以上内存

## CentOS7 关闭防火墙

|  |
| --- |
| //临时关闭  systemctl stop firewalld  //禁止开机启动  systemctl disable firewalld  Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.  Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service. |

# 负载均衡与反向代理

## 外网映射工具

在做微信开发或者是对接第三方支付接口时，回调接口可能需要外网访问。

这时候开发者在本地测试的时候，需要用到外网测试工具。

常用的外网测试工具有natapp、ngrok

### NatApp简介

服务器更新:全面支持HTTPS协议以及本地SSL证书,支持WSS协议.同时支持HTTP/2 WEB协议,支持微信小程序本地开发  
全面自动支持泛子域名与访客真实IP地址

Windows用法 natapp -authtoken=9ab6b9040a624f40

相关文档<https://natapp.cn/>

## 负载均衡

负载均衡 建立在现有网络结构之上，它提供了一种廉价有效透明的方法扩展网络设备和服务器的带宽、增加吞吐量、加强网络数据处理能力、提高网络的灵活性和可用性。

负载均衡，英文名称为Load Balance，其意思就是分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。

## Nginx

     Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件(IMAP/POP3)代理服务器，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev所开发，供俄国大型的入口网站及搜索引擎Rambler(俄文：Рамблер)使用。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好.中国大陆使用nginx网站用户有：新浪、网易、 腾讯等。

Nginx 是一个高性能的 Web 和反向代理服务器, 它具有有很多非常优越的特性:

作为 Web 服务器：相比 Apache，Nginx 使用更少的资源，支持更多的并发连接，体现更高的效率，这点使 Nginx 尤其受到虚拟主机提供商的欢迎。能够支持高达 50,000 个并发连接数的响应，感谢 Nginx 为我们选择了 epoll and kqueue 作为开发模型.

作为负载均衡服务器：Nginx 既可以在内部直接支持 Rails 和 PHP，也可以支持作为 HTTP代理服务器 对外进行服务。Nginx 用 C 编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好的多。

作为邮件代理服务器: Nginx 同时也是一个非常优秀的邮件代理服务器（最早开发这个产品的目的之一也是作为邮件代理服务器），Last.fm 描述了成功并且美妙的使用经验。

Nginx 安装非常的简单，配置文件 非常简洁（还能够支持perl语法），Bugs非常少的服务器: Nginx 启动特别容易，并且几乎可以做到7\*24不间断运行，即使运行数个月也不需要重新启动。你还能够在 不间断服务的情况下进行软件版本的升级。

Nginx一般用户七层负载均衡，其吞吐量有一定的限制。为了提高整体的吞吐量，会在DNS和Nginx之间引入LVS（软件负载均衡器）、F5（硬负载均衡器） 可以做四层负载均衡，首先DNS解析到LVS(F5)，让后LVS(F5)转发给Nginx，在有Nginx转发给真实的服务器

### Nginx基本安装

#### Windows安装Nginx

解压:nginx-windows

双击: nginx.exe



能看到nginx欢迎界面说明,nginx安装成功

演示下 nginx做静态服务器

#### Linux安装Nginx

1.安装gcc gcc-c++(如新环境,未安装请先安装)

|  |
| --- |
| $ yum install -y gcc gcc-c++ |

2.安装wget

|  |
| --- |
| $ yum -y install wget |

3.安装PCRE库

|  |
| --- |
| $ cd /usr/local/  $ wget http://jaist.dl.sourceforge.net/project/pcre/pcre/8.33/pcre-8.33.tar.gz  $ tar -zxvf pcre-8.33.tar.gz  $ cd pcre-8.33  $ ./configure  $ make && make install  如果报错:  在 linux 中执行 wget 命令提示 -bash: wget: command not found 解决方法  解决办法 yum -y install wget |

5.安装SSL库

|  |
| --- |
| $ cd /usr/local/  $ wget <http://www.openssl.org/source/openssl-1.0.1j.tar.gz>  $ tar -zxvf openssl-1.0.1j.tar.gz  $ cd openssl-1.0.1j  $ ./config  $ make && make install |

6.安装zlib库存

|  |
| --- |
| $ cd /usr/local/  $ wget http://zlib.net/zlib-1.2.11.tar.gz  $ tar -zxvf zlib-1.2.11.tar.gz  $ ./configure  $ make && make install |

5.安装nginx

|  |
| --- |
| $ cd /usr/local/  $ wget http://nginx.org/download/nginx-1.8.0.tar.gz  $ tar -zxvf nginx-1.8.0.tar.gz  $ cd nginx-1.8.0  $ ./configure  $ make && make install |

6.安装nginx

|  |
| --- |
| /usr/local/nginx/sbin/nginx |

### Nginx应用场景

1、http服务器。Nginx是一个http服务可以独立提供http服务。可以做网页静态服务器。

2、虚拟主机。可以实现在一台服务器虚拟出多个网站，例如个人网站使用的虚拟机。

3、反向代理，负载均衡。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用nginx做反向代理。并且多台服务器可以平均分担负载，不会应为某台服务器负载高宕机而某台服务器闲置的情况。

4、nginz 中也可以配置安全管理、比如可以使用Nginx搭建API接口网关,对每个接口服务进行拦截。

### Nginx目录结构

Nginx-

|\_ conf 配置目录

|\_ contrib

|\_ docs 文档目录

|\_ logs 日志目录

|\_ temp 临时文件目录

|\_ html 静态页面目录

|\_ nginx.exe 主程序

### Nginx静态资源

静态资源访问 存放在nginx的html页面

### Nginx虚拟主机配置

1、基于域名的虚拟主机，通过域名来区分虚拟主机——应用：外部网站

2、基于端口的虚拟主机，通过端口来区分虚拟主机——应用：公司内部网站，外部网站的管理后台

3、基于ip的虚拟主机，几乎不用。

#### 基于虚拟主机配置域名

实现步骤:

需要建立/data/www /data/bbs目录，windows本地hosts添加虚拟机ip地址对应的域名解析；对应域名网站目录下新增index.html文件；

|  |
| --- |
| #当客户端访问www.itmayiedu.com,监听端口号为80,直接跳转到data/www目录下文件  server {  listen 80;  server\_name www.itmayiedu.com;  location / {  root data/www;  index index.html index.htm;  }  }  #当客户端访问www.itmayiedu.com,监听端口号为80,直接跳转到data/bbs目录下文件  server {  listen 80;  server\_name bbs.itmayiedu.com;  location / {  root data/bbs;  index index.html index.htm;  }  } |

#### 基于端口的虚拟主机

使用端口来区分，浏览器使用域名或ip地址:端口号 访问

|  |
| --- |
| #当客户端访问www.itmayiedu.com,监听端口号为8080,直接跳转到data/www目录下文件  server {  listen 8080;  server\_name 8080.itmayiedu.com;  location / {  root data/www;  index index.html index.htm;  }  }    #当客户端访问www.itmayiedu.com,监听端口号为8081,直接跳转到data/bbs目录下文件  server {  listen 8081;  server\_name 8081.itmayiedu.com;  location / {  root data/bbs;  index index.html index.htm;  }  } |

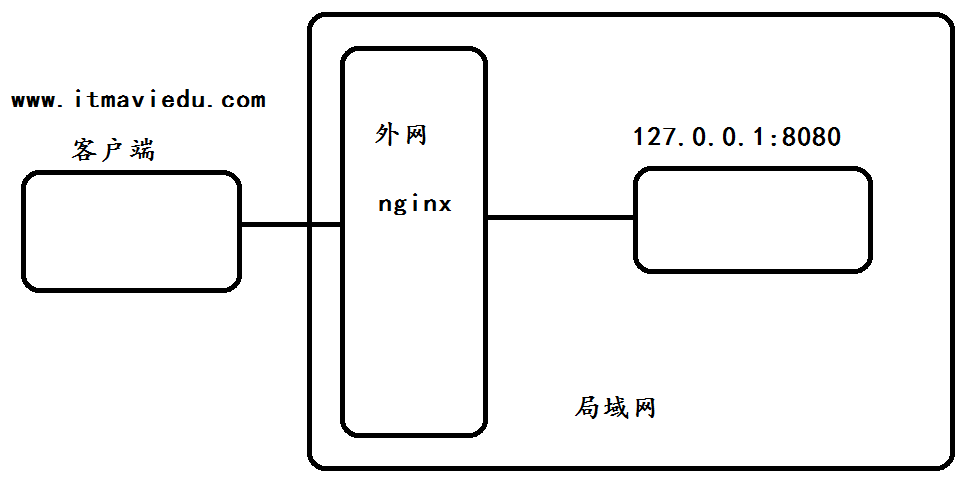
### Nginx配置反向代理

#### 反向代理的作用

反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

启动一个Tomcat 127.0.0.1:8080

使用nginx反向代理 8080.itmayiedu.com 直接跳转到127.0.0.1:8080



#### 反向代理的好处

反向代理的好处隐藏真实内部ip地址，请求先访问nginx代理服务器（外网可以访问到）,在使用nginx服务器转发到真实服务器中。

#### 反向代理的配置

|  |
| --- |
| ###当客户端访问www.itmayiedu.com,监听端口号为80直接跳转到真实ip服务器地址 127.0.0.1:8080  server {  listen 80;  server\_name www.itmayiedu.com;  location / {  proxy\_pass http://127.0.0.1:8080;  index index.html index.htm;  }  }  ###当客户端访问www.itmayiedu.com,监听端口号为80直接跳转到真实ip服务器地址 127.0.0.1:8081  server {  listen 80;  server\_name 8081.itmayiedu.com;  location / {  proxy\_pass http://127.0.0.1:8081;  index index.html index.htm;  }  } |

### Location正则表达式

#### location的作用

  location指令的作用是根据用户请求的URI来执行不同的应用，也就是根据用户请求的网站URL进行匹配，匹配成功即进行相关的操作。

#### location的语法

已=开头表示精确匹配  
如 A 中只匹配根目录结尾的请求，后面不能带任何字符串。

^~ 开头表示uri以某个常规字符串开头，不是正则匹配

~ 开头表示区分大小写的正则匹配;

~\* 开头表示不区分大小写的正则匹配

/ 通用匹配, 如果没有其它匹配,任何请求都会匹配到

#### Location正则案例

#精确匹配,/后面不能带任何字符

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  #精确匹配,注解后面不能带任何字符  location =/ {  proxy\_pass http://127.0.0.1:8080;  index index.html index.htm;  }  } |

#匹配所有以/开头请求

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  #匹配所有以/开头请求  location / {  proxy\_pass http://127.0.0.1:8080;  index index.html index.htm;  }  } |

### 以开头/itmayiedu\_8080拦截 默认开启不区分大小写

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  ### 以开头/itmayiedu\_8080 最终跳转到http://127.0.0.1:8080/;  location /itmayiedu\_8080/ {  proxy\_pass http://127.0.0.1:8080/;  index index.html index.htm;  }  ### 以开头/itmayiedu\_8080 最终跳转到http://127.0.0.1:8081/;  location /itmayiedu\_8081/ {  proxy\_pass http://127.0.0.1:8081/;  index index.html index.htm;  }  } |

### 开头区分大小写

### Upstream Server 负载均衡

Upstream Server 中文翻译 上游服务器，意思就是负载均衡服务器设置，白话文表示(就是被nginx代理最后真实访问的服务器)

负载均衡算法：配置多个上游服务器(真实业务逻辑访问的服务器)的负载均衡机制

失败重试机制:当上游服务器(真实业务逻辑访问的服务器)出现超时或者服务器不存活，是否考虑重试机制(补偿机制)

服务器心跳检测: 当上游服务器(真实业务逻辑访问的服务器),监控检测|心跳检测

#### Nginx配置负载均衡

Nginx负载均衡提供上游服务器(真实业务逻辑访问的服务器),负载均衡、故障转移、失败重试、容错、健康检查等。

当上游服务器(真实业务逻辑访问的服务器)发生故障时，可以转移到其他上游服务器(真实业务逻辑访问的服务器)。

#### Upstream Server配置

upstream 主要配置如下:

IP地址和端口号：配置上游服务器的IP地址和端口

|  |
| --- |
| ###定义上游服务器(需要被nginx真实代理访问的服务器) 默认是轮训机制  upstream backServer{  server 127.0.0.1:8080;  server 127.0.0.1:8081;  }    server {  listen 80;  server\_name www.itmayiedu.com;  location / {  ### 指定上游服务器负载均衡服务器  proxy\_pass http://backServer;  index index.html index.htm;  }  } |

#### 负载均衡算法

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务，如果后端某台服务器死机，自动剔除故障系统，使用户访问不受影响。

2、weight（轮询权值）

weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。或者仅仅为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。

3、ip\_hash

每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题。俗称IP绑定。

4、fair（第三方）

比 weight、ip\_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间 来分配请求，响应时间短的优先分配。Nginx本身不支持fair，如果需要这种调度算法，则必须安装upstream\_fair模块。

5、url\_hash(第三方)

按访问的URL的哈希结果来分配请求，使每个URL定向到一台后端服务器，可以进一步提高后端缓存服务器的效率。Nginx本身不支持url\_hash，如果需要这种调度算法，则必须安装Nginx的hash软件包。

##### 轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务，如果后端某台服务器死机，自动剔除故障系统，使用户访问不受影响。

##### 权重Weight

|  |
| --- |
| upstream backServer{  server 127.0.0.1:8080 weight=1;  server 127.0.0.1:8081 weight=2;  }    server {  listen 80;  server\_name www.itmayiedu.com;  location / {  ### 指定上游服务器负载均衡服务器  proxy\_pass http://backServer;  index index.html index.htm;  }  } |

##### IP绑定ip\_hash

每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题。俗称IP绑定。

|  |
| --- |
| upstream backServer{  server 127.0.0.1:8080 ;  server 127.0.0.1:8081 ;  ip\_hash;  }    server {  listen 80;  server\_name www.itmayiedu.com;  location / {  ### 指定上游服务器负载均衡服务器  proxy\_pass http://backServer;  index index.html index.htm;  }  } |

### Nginx配置故障转移

当上游服务器(真实访问服务器),一旦出现故障或者是没有及时相应的话，应该直接轮训到下一台服务器，保证服务器的高可用。

Nginx配置代码

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  location / {  ### 指定上游服务器负载均衡服务器  proxy\_pass http://backServer;  ###nginx与上游服务器(真实访问的服务器)超时时间 后端服务器连接的超时时间\_发起握手等候响应超时时间  proxy\_connect\_timeout 1s;  ###nginx发送给上游服务器(真实访问的服务器)超时时间  proxy\_send\_timeout 1s;  ### nginx接受上游服务器(真实访问的服务器)超时时间  proxy\_read\_timeout 1s;  index index.html index.htm;  }  } |

### Nginx配置失败重试策略

### Nginx配置健康检查

### nginx rewrite

### Nginx开启Gzip压缩

### Nginx配置动静分离

### Nginx信号量

Nginx中使用信号来控制Nginx停止、平滑重启，Nginx支持以下几种信号：

TERM,INT :kill -TERM（INT） nginx主进程号或者kill -TERM（INT） `cat/usr/local/nginx/logs/nginx.pid `，nginx的进程马上被关闭，不能完整处理正在使用的nginx的用户的请求，等同于 /usr/local/nginx -s stop

QUIT:kill -QUIT nginx主进程号 优雅的关闭nginx进程，在处理完所有正在使用nginx用户请求后再关闭nginx进程，等同于/usr/local/nginx -s quit

HUP:kill -HUP nginx主进程号 nginx进程不关闭，但是重新加载配置文件。等同于/usr/local/nginx -s reload

USR1:kill -USR1 nginx主进程号 不用关闭nginx进程就可以重读日志，此命令可以用于nginx的日志定时备份，按月/日等时间间隔分割有用

USR2:kill -USR2 nginx主进程号 nginx的版本需要升级的时候，不需要停止nginx，就能对nginx升级

WINCH:kill -WINCH nginx主进程号 配合USR2对nginx升级，优雅的关闭nginx旧版本的进程。

### Nginx日志管理

## Http动态负载均衡

## lvs+keepalived+nginx实现高性能负载均衡集群

# 高并发服务限流特技

# 高并发服务降级特技

# 常见名称

高并发

高可用

幂等性

心跳检测

隔离技术

限流技术

降级技术

雪崩效应

超时机制

防重设计

重试机制

补偿机制

回滚机制

注册中心

服务化

拆分

命中率

多级缓存

异步并发

平滑扩容

可伸缩

敏捷迭代

无状态