Nginx

高并发下的Nginx性能优化实战

讲师：李康

课程目标

1. Nginx入门
2. Nginx实战应用 

3. 高并发下的Nginx优化 

解读Nginx的核心知识、掌握nginx核心原理、 Nginx在高并发下的性能优化

1. Nginx入门

1. Nginx背景
2. Nginx概述
3. Nginx优势
4. Nginx功能
5. Linux下Nginx安装
6. Linux下Nginx命令

2.Nginx实战应用

1. Nginx配置文件详解
2. 代理模式
3. Nginx配置web应用集群搭建
4. Nginx负载均衡
5. Nginx配置日志
6. Nginx动静分离(缓存)
7. 热部署

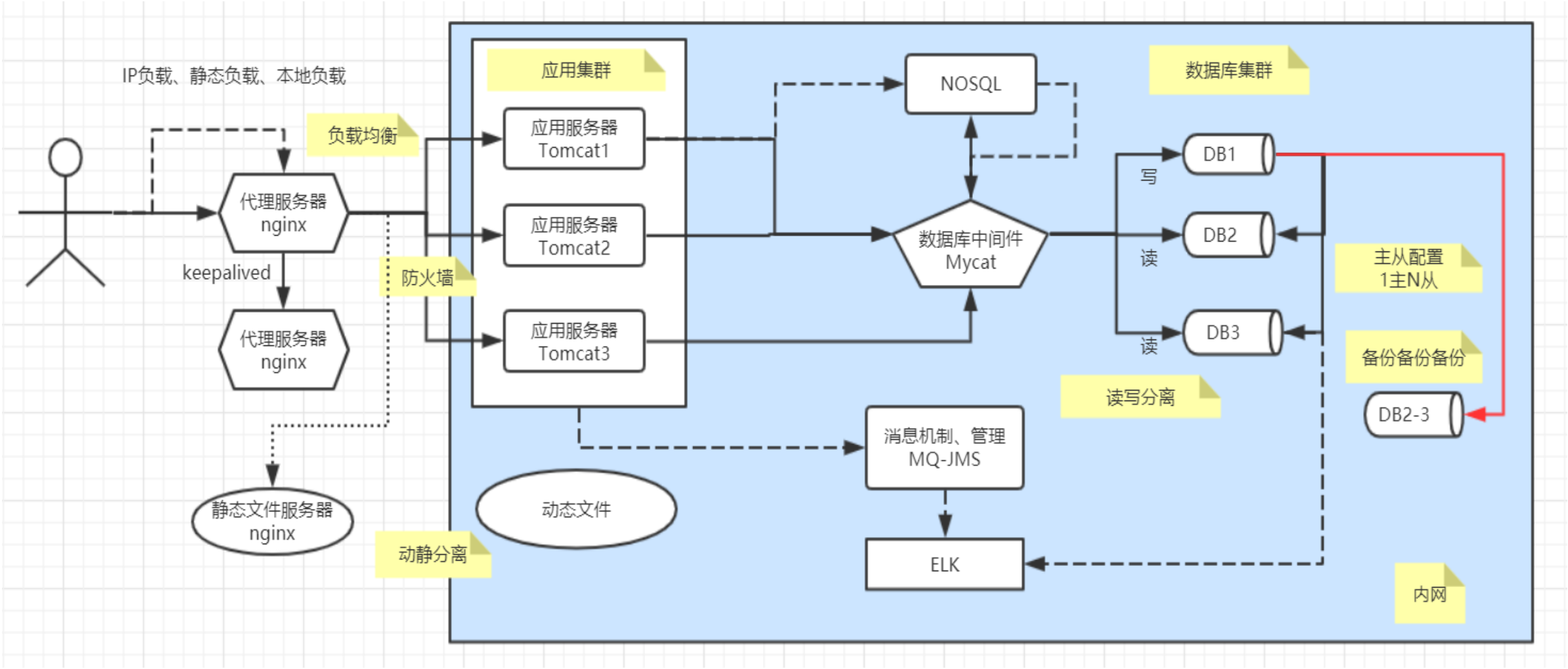
3.高并发下Nginx优化

1. 高并发架构分析
2. 高并发下Nginx配置限流
3. 高并发下Nginx安全配置
4. Nginx配置优化之进程数、并发连接数
5. Nginx配置优化之长连接
6. Nginx配置优化之压缩
7. Nginx配置优化之系统优化
8. Nginx配置优化之状态监控
9. Nginx与其它解决方案搭配组合

收益

1. 通过本课程的学习，将可以掌握Nginx在Linux系统下的配置使用
2. 可以掌握高并发下的Nginx性能优化
3. 通过课程学习，将所学能够快速融合到企业项目中

企业项目架构部署


IP负载、静态负载、本地负载
负载均衡
代理服务器
nginx
keepalived
代理服务器
nginx
防火墙
静态文件服务器
nginx
应用集群
应用服务器
Tomcat1
应用服务器
Tomcat2
应用服务器
Tomcat3
NOSQL
数据库中间件
Mycat
数据库集群
写
读
读
DB1
DB2
DB3
主从配置
1主N从
备份备份备份
DB2-3
读写分离
消息机制、管理
MQ-JMS
ELK
动态文件
动静分离
内网

1. Nginx入门-本章目标

1. Nginx概述 

2. Linux下Nginx安装
3. Linux下Nginx命令 

Nginx概述-本节目标

1. 掌握Nginx功能、应用场景

什么是Nginx-背景

Nginx同Apache一样都是一种WEB服务器。基于REST架构风格，以统一资源描述符(Uniform Resources Identifier)URI或者统一资源定位符(Uniform Resources Locator)URL作为沟通依据，通过HTTP协议提供各种网络服务 。

然而，这些服务器在设计之初受到当时环境的局限，例如当时的用户规模，网络带宽，产品特点等局限并且各自的定位和发展都不尽相同。这也使得各个WEB服务器有着各自鲜明的特点 。

Apache的发展时期很长，而且是毫无争议的世界第一大服务器。它有着很多优点：稳定、开源、跨平台等等。它出现的时间太长了，它兴起的年代，互联网产业远远比不上现在。所以它被设计为一个重量级的。它不支持高并发的服务器。在Apache上运行数以万计的并发访问，会导致服务器消耗大量内存。操作系统对其进行进程或线程间的切换也消耗了大量的CPU资源，导致HTTP请求的平均响应速度降低

这些都决定了Apache不可能成为高性能WEB服务器，轻量级高并发服务器Nginx就应运而生了。

什么是Nginx-概述

- 是一个高性能的HTTP和反向代理web服务器，轻量级
- 提供了IMAP/POP3/SMTP服务
- 发布于2004年10月4日（第一个公开版本0.1.0 ）
- Nginx 的1.4.0稳定版已经于2013年4月24日发布
- C语言编写
- Nginx是一个跨平台服务器
- Nginx有自己的函数库，并且除了zlib、PCRE和OpenSSL之外，标准模块只使用系统C库函数。而且，如果不需要或者考虑到潜在的授权冲突，可以不使用这些第三方库

什么是Nginx-优势

- 占有内存少 (在3W并发连接中，开启的10个nginx进程消耗内存大约150M)
- 高并发能力强(官方测试能够支撑5W并发连接，在实际生产环境中能到2-3W并发连接数)
- 简单(配置文件通俗易懂)
- 价格(免费、开源)
- 支持Rewriter重写(能够根据域名、URL的不同，将HTTP请求分到不同的后端服务器群组)
- 内置健康检查(如果nginx后端有几个服务宕机了，不会影响前端访问，能自动检测服务状态)
- 节省带宽(支持GZIP压缩，可以添加浏览器本地缓存的Header头)
- 稳定性高，反向代理，很少宕机
- 中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等

什么是Nginx-应用场景

功能

- web服务器、轻量级
- 负载、均衡
- 缓存
- 高并发

应用场景

- 代理服务器
- IP负载、静态负载
- 动静分离
- 限流、健康监控

2.Linux下Nginx安装

安装步骤：

```
yum install gcc-c++
yum -y install pcre pcre-devel
yum -y install zlib zlib-devel
yum install -y openssl openssl-devel


wget  http://nginx.org/download/nginx-1.13.11.tar.gz
tar zxvf nginx-1.13.11.tar.gz
./configure  --prefix=/opt/nginx
make
make install
```

平台环境：

虚拟机：VMware Workstation

系　统：Linux：CentOS-7-x86_64

远程连接工具：

CRT(SSH工具)

3.Linux下Nginx命令

命令：

cd sbin/

./nginx
./nginx -s stop
./nginx -s quit
./nginx -s reload

./nginx -s quit：此方式停止步骤是待nginx进程处理任务完毕进行停止。
./nginx -s stop：此方式相当于先查出nginx进程id再使用kill命令强制杀掉进程。

注意：

开启端口： /sbin/iptables -I INPUT -p tcp --dport 80 -j ACCEPT
如果本地访问不到端口，则需要开启
如果开启之后，还访问不到，则需要关闭本地防火墙

2.Nginx实战应用

1. Nginx配置文件详解
2. 代理模式
3. Nginx配置web应用集群搭建
4. Nginx负载均衡
5. Nginx配置日志
6. Nginx动静分离(缓存)
7. 热部署

Nginx配置文件详解

#模块结构

- 核心模块
    1. HTTP 模块(代理、缓存、日志定义和第三方模块)
    2. EVENTS 模块(网络连接)
    3. 全局 模块(全局指令，日志路径、PID路径、用户信息等)

- 基础模块
    1. HTTP 全局 模块
    2. HTTP FastCGI 模块
    3. HTTP Gzip模块
    4. HTTP server模块(虚拟主机，一个http，可以有多个server)
    5. HTTP location 模块(请求的路由，各种页面的处理)
    6. HTTP Rewrite模块

- 第三方模块
    1. HTTP Upstream Request Hash 模块
    2. Notice 模块
    3. HTTP Access Key模块

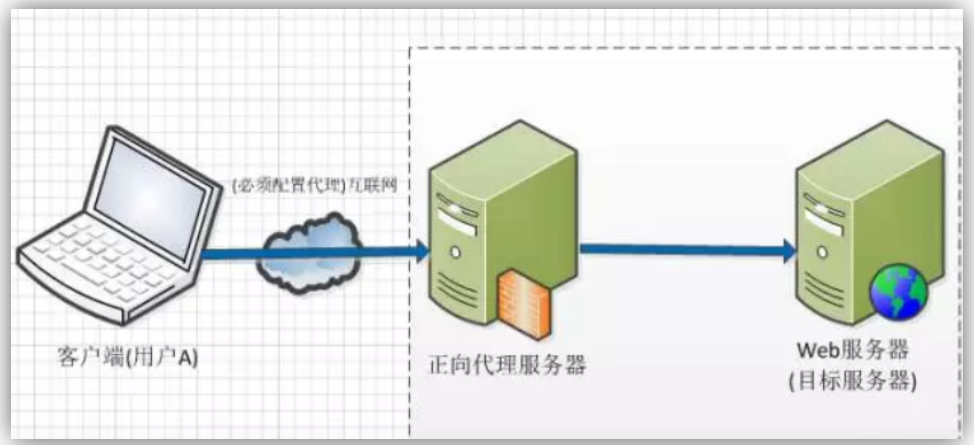
Nginx配置文件详解

```
...              #全局块
events {         #events块
   ...
}
http      #http块
{
    ...   #http全局块
    gzip\upstream\fastcgi
    server        #server块
    {
        ...       #server全局块
        location [PATTERN]   #location块
        {
            ...
        }
        location [PATTERN]
        {
            ...
        }
    }
    server
    {
      ...
    }
    ...     #http全局块
}
```

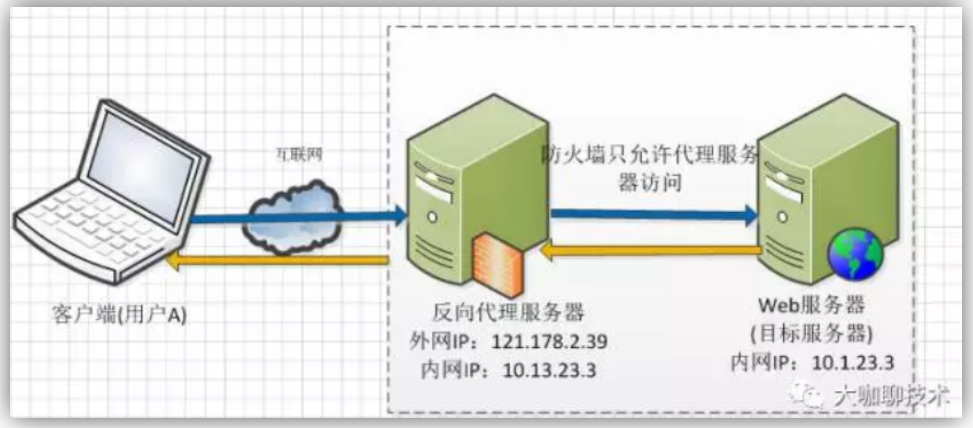
Nginx代理模式

#代理模式

- 正向代理

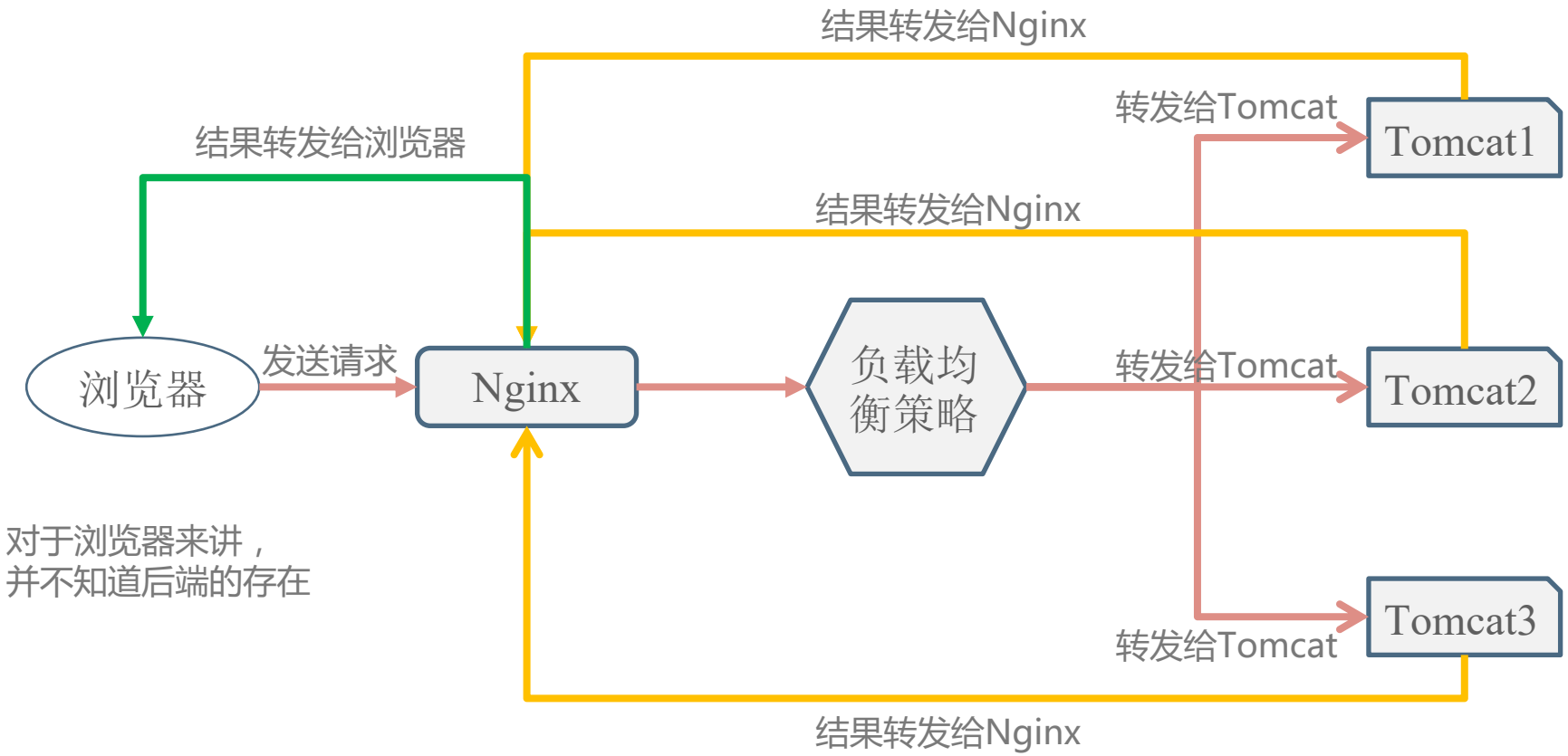


- 反向代理



- 透明代理

Nginx集群搭建



对于Tomcat来说，
当前客户端为Nginx

Nginx负载均衡

轮询法(默认)

- 将请求按顺序轮流地分配到后端服务器上，它均衡地对待后端的每一台服务器，而不关心服务器实际的连接数和当前的系统负载

```
upstream tomcat_server {
    server 192.168.10.11:8080 weight=1;
    server 192.168.10.12:8080 weight=1;
}
```

加权轮询法(weight)

- 不同的后端服务器可能机器的配置和当前系统的负载并不相同，因此它们的抗压能力也不相同
- 给配置高、负载低的机器配置更高的权重，让其处理更多的请求
- 配置低、负载高的机器，给其分配较低的权重，降低其系统负载
- 加权轮询能很好地将请求顺序且按照权重分配到后端
- weight的值越大分配到的访问概率越高

```
upstream tomcat_server {
    server 192.168.10.11:8080 weight=1;
    server 192.168.10.12:8080 weight=2;
}
```

Nginx负载均衡

源地址哈希法

- 根据获取客户端的IP地址，通过哈希函数计算得到一个数值
- 用该数值对服务器列表的大小进行取模运算，得到的结果便是客服端要访问服务器的序号
- 采用源地址哈希法进行负载均衡，同一IP地址的客户端，当后端服务器列表不变时，它每次都会映射到同一台后端服务器进行访问
- 可以保证来自同一ip的请求被打到固定的机器上，可以解决session问题

```
upstream tomcat_server {
   ip_hash;
   server 192.168.10.11:8080 weight=1;
   server 192.168.10.12:8080 weight=1;
}
```

最小连接数法（least_conn）

- 由于后端服务器的配置不尽相同，对于请求的处理有快有慢，最小连接数法根据后端服务器当前的连接情况，动态地选取其中当前积压连接数最少的一台服务器来处理当前的请求，尽可能地提高后端服务的利用效率，将负责合理地分流到每一台服务器

```
upstream tomcat_server {
   least_conn;
   server 192.168.10.11:8080 weight=1;
   server 192.168.10.12:8080 weight=1;
}
```

Nginx负载均衡

Fair

- 比weight、ip_hash更智能的负载均衡算法
- 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upsteram_fair模块

```
upstream tomcat_server {
    fair;
   server 192.168.10.11:8080 weight=1;
   server 192.168.10.12:8080 weight=1;
}
```

url_hash

- 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

```
upstream tomcat_server {
   hash $request_uri;
   server 192.168.10.11:8080 weight=1;
   server 192.168.10.12:8080 weight=1;
}
```

Nginx负载均衡

Fair插件安装

- 下载地址：https://github.com/gnosek/nginx-upstream-fair/tree/master
- 解压zip：unzip nginx-upstream-fair-master.zip
- 增加模块： ./configure --prefix=/opt/nginx --add-module=/opt/nginx-upstream-fair-master
- defalut_port问题修改：cd nginx-upstream-fair-master
- sed -i 's/default_port/no_port/g' ngx_http_upstream_fair_module.c
- make
- make install

hash插件安装

- 下载地址：https://github.com/evanmiller/nginx_upstream_hash
- 解压zip：unzip nginx_upstream_hash-master.zip
- 增加模块： ./configure --prefix=/opt/nginx --add-module=/opt/ nginx_upstream_hash-master
- make
- make install

Nginx配置日志

- 通过访问日志，你可以得到用户地域来源、跳转来源、使用终端、某个URL访问量等相关信息
- 通过错误日志，你可以得到系统某个服务或server的性能瓶颈等
- 日志生成的到根目录logs/access.log文件，默认使用“main”日志格式，也可以自定义格式
- 默认“main”日志格式

```
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                  '$status $body_bytes_sent "$http_referer" '
                  '"$http_user_agent" "$http_x_forwarded_for"';
```

$remote_addr：	客户端的ip地址(代理服务器，显示代理服务ip)
$remote_user：	用于记录远程客户端的用户名称（一般为“-”）
$time_local：	用于记录访问时间和时区
$request：	用于记录请求的url以及请求方法
$status：	响应状态码，例如：200成功、404页面找不到等。
$body_bytes_sent：	给客户端发送的文件主体内容字节数
$http_user_agent：	用户所使用的代理（一般为浏览器）
$http_x_forwarded_for：	可以记录客户端IP，通过代理服务器来记录客户端的ip地址
$http_referer：	可以记录用户是从哪个链接访问过来的

Nginx动静分离

- Nginx 动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离
- 严格意义上说应该是动态请求跟静态请求分开，可以理解成使用Nginx 处理静态页面，Tomcat处理动态页面
- 动静分离从目前实现角度来讲大致分为两种

1. 一种是纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案
2. 一种是动态跟静态文件混合在一起发布，通过 nginx 来分开

Nginx热部署

1：Nginx版本升级、模块修订

2：后台服务器Tomcat问题修订

3.高并发下Nginx优化

1. 高并发架构分析
2. 高并发下Nginx配置限流
3. 高并发下Nginx安全配置
4. Nginx配置优化之进程数、并发连接数、系统优化
5. Nginx配置优化之长连接
6. Nginx配置优化之压缩
7. Nginx配置优化之状态监控
8. Nginx与其它解决方案搭配组合

3.1高并发架构分析

什么是高并发?

- 高并发（High Concurrency）是互联网分布式系统架构设计中必须考虑的因素之一，它通常是指，通过设计保证系统能够同时并行处理很多请求。
- 高并发相关常用的一些指标有响应时间（Response Time），吞吐量（Throughput），每秒查询率QPS（Query Per Second），并发用户数等。
    - 响应时间：系统对请求做出响应的时间
    - 吞吐量：单位时间内处理的请求数量。
    - QPS：每秒响应请求数

3.1高并发架构分析

如何提升系统的并发能力?

- 互联网分布式架构设计，提高系统并发能力的方式，方法论上主要有两种：垂直扩展（Scale Up）与水平扩展（Scale Out）。
- 垂直扩展：提升单机处理能力。垂直扩展的方式又有两种。
  - 增强单机硬件性能
  - 提升单机架构性能

在互联网业务发展非常迅猛的早期，如果预算不是问题，强烈建议使用“增强单机硬件性能”的方式提升系统并发能力，因为这个阶段，公司的战略往往是发展业务抢时间，而“增强单机硬件性能”往往是最快的方法。

不管是提升单机硬件性能，还是提升单机架构性能，都有一个致命的不足：单机性能总是有极限的。所以互联网分布式架构设计高并发终极解决方案还是水平扩展。

水平扩展：只要增加服务器数量，就能线性扩充系统性能。

3.2高并发下Nginx配置限流

三种方式实现：

- limit_conn_zone
- limit_req_zone
- ngx_http_upstream_module

前两种只能对客户端（即单一ip限流）

3.2高并发下Nginx配置限流

Ab工具：

- Centos安装：yum install httpd-tools –y
- 测试：ab -c 10 -n 1000 http://www.test.com/

key	含义
Document Path	测试的页面
Document Length	页面的大小
Concurrency Level	并发数量、**并发用户数**
Time taken for tests	测试耗费总时间
Complete requests	请求总量、**并发连接数**
Failed requests	请求失败的数量
Write errors	错误数量
Requests per second	每秒钟的请求量、**吞吐率**
Time per request	每次请求需要时间、**响应时间**

3.2高并发下Nginx配置限流

- limit_conn_zone

```
http{
 limit_conn_zone $binary_remote_addr zone=one:10m;
 server
 {
   ......
  limit_conn one 10;
  ......
 }
}
```

- 其中“limit_conn one 10”既可以放在server层对整个server有效，也可以放在location中只对单独的location有效
- 该配置表明：客户端的并发连接数只能是10个。

3.2高并发下Nginx配置限流

- limit_req_zone

```
http{
 limit_req_zone $binary_remote_addr zone=req_one:10m rate=1r/s;
 server
 {
   ......
  limit_req zone=req_one burst=120;
  ......
 }
}
```

- 其中“limit_req zone=req_one burst=120”既可以放在server层对整个server有效，也可以放在location中只对单独的location有效
- rate=1r/s的意思是每个地址每秒只能请求一次，也就是说令牌桶burst=120一共有120块令牌，并且每秒钟只新增1块令牌，120块令牌发完后，多出来的请求就会返回503

3.2高并发下Nginx配置限流

- ngx_http_upstream_module(推荐)
  - 该模块是提供了我们需要的后端限流功能的
  - 该模块有一个参数：max_conns可以对服务端进行限流，可惜在商业版nginx中才能使用
  - 在nginx1.11.5版本以后，官方已经将该参数从商业版中脱离出来了，也就是说只要我们将生产上广泛使用的nginx1.9.12版本和1.10版本升级即可使用

```
upstream xxxx{
 server 127.0.0.1:8080 max_conns=10;
 server 127.0.0.1:8081 max_conns=10;
}
```

3.3高并发下Nginx安全配置

- 版本安全

```
http { server_tokens off; }
```

- IP安全

```
白名单配置：
location / {
    allow   192.168.136.1;
    deny    all;
}
```

```
黑名单设置：
location / {
    deny   192.168.136.1;
    allow    all;
}
```

- 文件安全

```
location /logs {
      autoindex on;
      root /opt/nginx/;
}
```

```
location ^/logs~*\.(log|txt)$ {
              add_header Content-Type text/plain;
              root /opt/nginx/;
}
```

- 连接安全

```
HTTPS开启
```

3.4Nginx配置进程数、并发数、系统优化

- 调整Nginx的主配置文件,增加并发量

```
worker_processes 2;      #调整到与CPU数量一致
events {
        worker_connection 65535;   #每个worker最大并发连接数
}
```

- 调整内核参数

```
[root@proxy ~]# ulimit -a    #查看所有的属性值
[root@proxy ~]# ulimit -Hn 100000    #临时设置硬限制
[root@proxy ~]# ulimit -Sn 100000    #设置软限制
[root@proxy ~]# vim /etc/security/limits.conf
...
*               soft        nofile                  100000
*               hard        nofile                  100000

用户/组     软/硬限制     需要限制的项目        限制的值
```

- 验证

```
[root@proxy ~]# ab -n 2000 -c 2000 http://192.168.136.131/      #自己访问自己,测试一下配置效果
```

3.5高并发下Nginx长连接

- nginx长连接短连接，可以增强服务器的容灾能力
- 场景：

HTTP1.1之后，HTTP协议支持持久连接，也就是长连接，优点在于在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。如果我们使用了nginx去作为反向代理或者负载均衡，从客户端过来的长连接请求就会被转换成短连接发送给服务器端，为了支持长连接，我们需要在nginx服务器上做一些配置

- 要求：

使用nginx时，想要做到长连接，我们必须做到以下两点：

1：从client到nginx是长连接

2：从nginx到server是长连接

对于客户端而言，nginx其实扮演着server的角色，反之，之于server，nginx就是一个client

- 配置：

我们要想做到Client与Nginx之间保持长连接，需要：

1：Client发送过来的请求携带"keep-alive"header。

2：Nginx设置支持keep-alive

3.6高并发下Nginx压缩

- gzip压缩作用：将响应报文发送至客户端之前可以启用压缩功能，这能够有效地节约带宽，并提高响应至客户端的速度,压缩会消耗nginx的cpu性能。
- gzip压缩可以配置http,server和location模块下

3.7高并发下Nginx状态监控

配置Nginx的监控选项(配置文件路径：nginx.conf)

添加如下代码：

```
#设定Nginx状态访问地址
location /NginxStatus {
        stub_status on;
        access_log off;
    }
```

插件安装：./configure --prefix=/opt/nginx/ --with-http_stub_status_module

配置完成重启Nginx

配置完成后在浏览器中输入http://192.168.136.131/NginxStatus查看

3.7高并发下Nginx状态监控

参数说明：

#活跃的连接数量

active connections

#总共处理了n个连接，成功创建n次握手, 总共处理了n个请求

server accepts handled requests

#每个连接有三种状态waiting、reading、writing

reading —读取客户端的Header信息数.这个操作只是读取头部信息，读取完后马上进入writing状态，因此时间很短

writing — 响应数据到客户端的Header信息数.这个操作不仅读取头部，还要等待服务响应，因此时间比较长。

waiting — 开启keep-alive后等候下一次请求指令的驻留连接.

正常情况下waiting数量是比较多的，并不能说明性能差。反而如果reading+writing数量比较多说明服务并发有问题。

查看Nginx并发进程数：ps -ef|grep nginx | wc -l

查看Web服务器TCP连接状态：netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

3.7高并发下Nginx状态监控

解析：

CLOSED //无连接是活动的或正在进行

LISTEN //服务器在等待进入呼叫

SYN_RECV //一个连接请求已经到达，等待确认

SYN_SENT //应用已经开始，打开一个连接

ESTABLISHED //正常数据传输状态/当前并发连接数

FIN_WAIT1 //应用说它已经完成

FIN_WAIT2 //另一边已同意释放

ITMED_WAIT //等待所有分组死掉

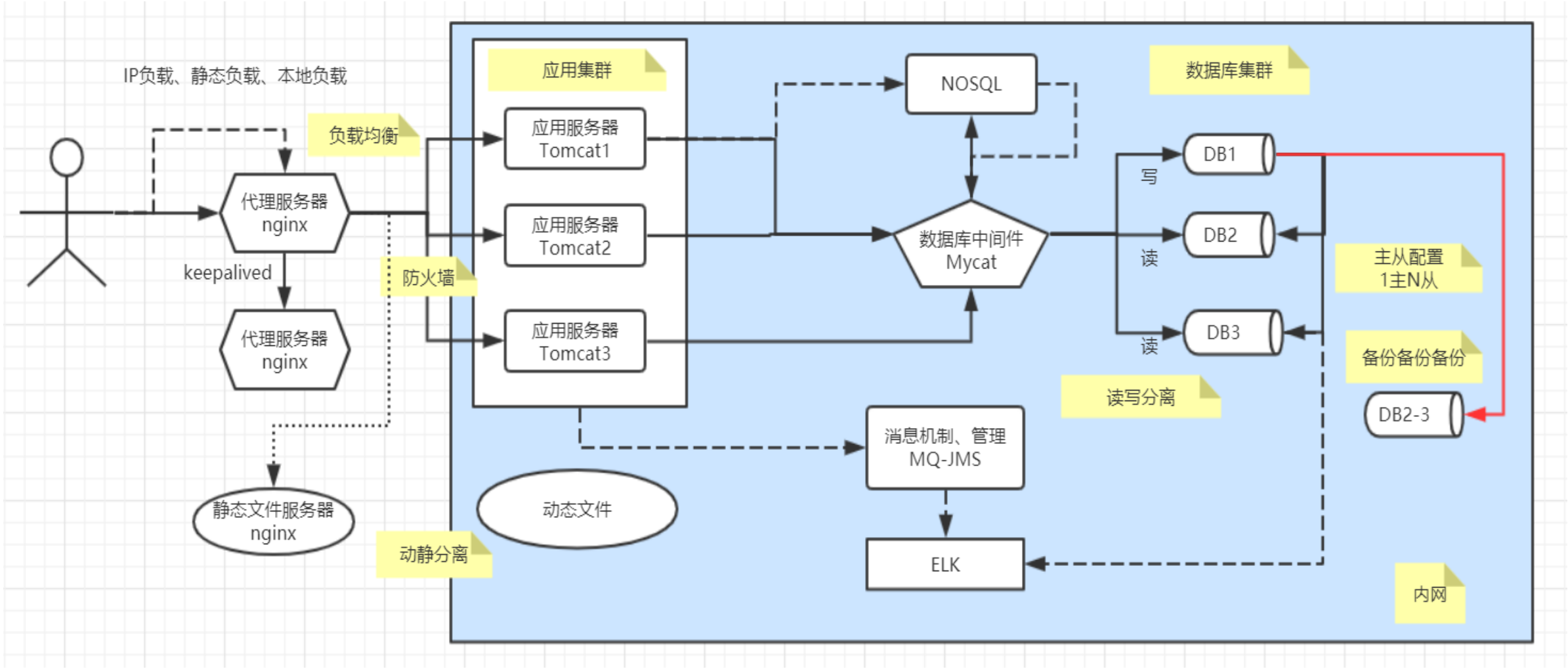
CLOSING //两边同时尝试关闭

TIME_WAIT //另一边已初始化一个释放

LAST_ACK //等待所有分组死掉

查看Web服务器TCP连接状态：netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

3.8高并发下Nginx整合方案



总结

1. 通过本课程的学习，将可以掌握Nginx在Linux系统下的配置使用
2. 可以掌握高并发下的Nginx性能优化
3. 通过课程学习，将所学能够快速融合到企业项目中

EDU

CSDN学院　IT实战派