**目 录**

**[1](#_Toc267644488)****[网站性能瓶颈分析 1](#_Toc267644488)**

**[2](#_Toc267644490)****[系统架构设计 2](#_Toc267644490)**

**[2.1](#_Toc267644491)** [总体思路 2](#_Toc267644491)

2.1.1 WEB动态分离 2

2.1.2 数据存储思路 2

2.1.3 不同用户网络考虑 2

**[2.2](#_Toc267644492)** [总体机构 2](#_Toc267644492)

2.2.1 网站的系统架构 2

2.2.2 服务器硬件配置 3

2.2.3 服务应用配置 4

**[3](#_Toc267644504)****[硬件扩容策略 10](#_Toc267644504)**

**[3.1](#_Toc267644505)** [增加服务器 10](#_Toc267644505)

**[3.2](#_Toc267644506)** [增加存储 10](#_Toc267644506)

**[3.3](#_Toc267644509)** [升级服务器 11](#_Toc267644509)

**[3.4](#_Toc267644510)** [网络扩容 11](#_Toc267644510)

**[3.5](#_Toc267644511)** [网络架构升级 11](#_Toc267644511)

# 1 . 网站的性能瓶颈分析

网站的性能影响因素很多，下面主要从如下4个方面进行分析说明：

1. 网络负载
2. 公网负载
3. 内网负载
4. WEB应用服务器性能
   1. CPU
   2. 存储，I/O访问
   3. 内存
   4. 并发TCP/IP连接数
5. 数据库服务器性能
   1. 数据库参数配置
   2. 服务器性能（CPU、内存、存储）
   3. 数据结构的合理性
6. 不同WEB应用的处理方式而对不同的性能瓶颈
   1. 对于静态的网站：

静态的HTML页面严格地由标准的HTML标示语言构成，并不需要服务器端即时运算生成。这意味着，对一个静态HTML文档发出访问请求后，服务器端只是简单地将该文档传输到客户端。从服务器运行的那个时间片来看，这个传输过程仅仅占用了很小的CPU资源。对于静态HTML的访问瓶颈为：网络带宽、磁盘I/O以及cache(高速缓冲存储器)。

* 1. 对于动态页面

因为服务器解析动态页面必须在其传输到客户端前就通过服务器来进行解释，这样就会给应用服务器添加额外的性能消耗，如果进一步要访问数据库，则会增加数据库服务器的性能消耗，则动态页面还有额外的瓶颈：应用服务器的性能，数据库服务器的性能。

# 2. 系统架构设计

## 2.1总体思路

为提高网站的高并发性能,提高开发效率及运营效率,主要按如下几个思路进行规划设计:

### 2.1.1 WEB动静分离

1. Nginx

通过Nginx去处理静态html、图片等。

1. Tomcat

通过Tomcat处理动态数据。

### 2.1.2 数据存储思路

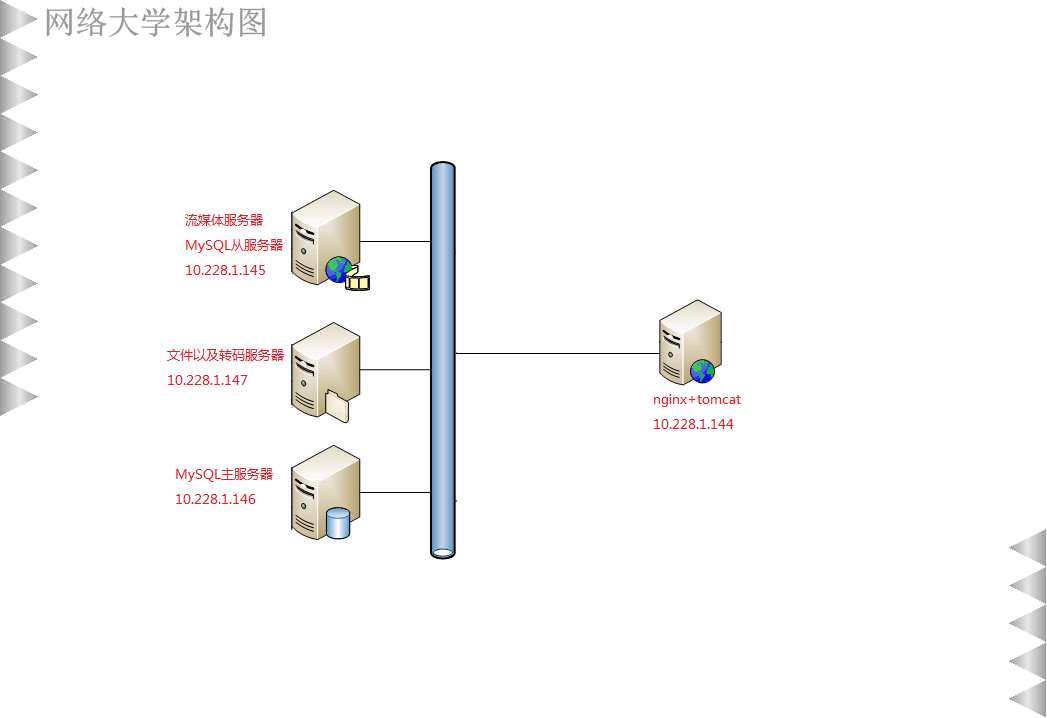
1. 采用mysql主备方式，当备数据库出现问题马上切找到备库。
2. 采用单独服务器存储流媒体数据，通过wowa进行。

### 2.1.3 不同网络用户访问考虑

* 1. 通过引入CDN来解决不同网络服务商的接入速度问题，一般只能解决静态页面的访问问题。
  2. 在不同运营商机房部署服务器，通过镜像技术来实现不同网络服务商的接入速度问题。

## 总体架构

### 2.2.1网站的系统架构



Nginx (“engine x”) 是俄罗斯人Igor Sysoev(塞索耶夫)编写的一款高性能的 HTTP 和反向代理服务器。

Nginx 已经在俄罗斯最大的门户网站── Rambler Media（www.rambler.ru）上运行了4年时间，同时俄罗斯超过20%的虚拟主机平台采用Nginx作为反向代理服务器。

Nginx 特点如下：

1. 工作在OSI模型的第7层（应用层）
2. 高并发连接

官方测试能够支撑5万并发连接，在实际生产环境中跑到2～3万并发连接数。

1. 内存消耗少

在3万并发连接下，开启的10个Nginx 进程才消耗150M内存（15M\*10=150M）。

1. 配置文件非常简单

风格跟程序一样通俗易懂。

1. 成本低廉

Nginx为开源软件，可以免费使用。而购买F5 BIG-IP、NetScaler等硬件负载均衡交换机则需要十多万至几十万人民币。

1. 支持Rewrite重写规则

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。

1. 内置的健康检查功能

如果 Nginx Proxy 后端的某台 Web 服务器宕机了，不会影响前端访问。

1. 节省带宽

支持 GZIP 压缩，可以添加浏览器本地缓存的 Header 头。

1. 稳定性高

用于反向代理，宕机的概率微乎其微。

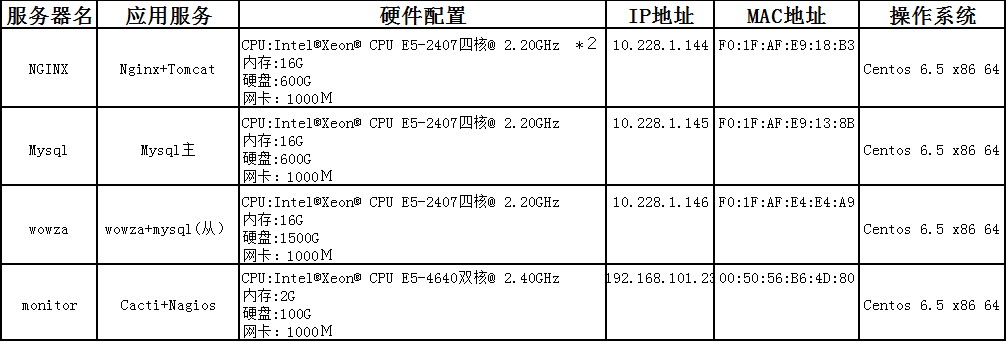
Tomcat:

术先进、性能稳定；

占用系统资源少，扩展性好；

配置简单，容易上手，使用人群广泛、熟悉程度最高。

### 2.2.2 服务器硬件配置



2.2.3 服务应用配置

1）LINUX优化：/etc/sysctl.conf

net.ipv4.ip\_forward = 0

net.ipv4.conf.default.rp\_filter = 1

net.ipv4.conf.default.accept\_source\_route = 0

kernel.sysrq = 0

kernel.core\_uses\_pid = 1

net.ipv4.tcp\_syncookies = 1

kernel.msgmnb = 65536

kernel.msgmax = 65536

kernel.shmmax = 68719476736

kernel.shmall = 4294967296

net.ipv4.tcp\_max\_tw\_buckets = 30000

net.ipv4.tcp\_sack = 1

net.ipv4.tcp\_window\_scaling = 1

net.ipv4.tcp\_rmem = 4096 87380 4194304

net.ipv4.tcp\_wmem = 4096 16384 4194304

net.nf\_conntrack\_max=6553600

net.netfilter.nf\_conntrack\_max=6553600

net.core.wmem\_default = 8388608

net.core.rmem\_default = 8388608

net.core.rmem\_max = 16777216

net.core.wmem\_max = 16777216

net.core.netdev\_max\_backlog = 262144

net.core.somaxconn = 262144

net.ipv4.tcp\_max\_orphans = 3276800

net.ipv4.tcp\_max\_syn\_backlog = 262144

net.ipv4.tcp\_timestamps = 0

net.ipv4.tcp\_synack\_retries = 1

net.ipv4.tcp\_syn\_retries = 1

net.ipv4.tcp\_tw\_recycle = 1

net.ipv4.tcp\_tw\_reuse = 1

net.ipv4.tcp\_mem = 94500000 915000000 927000000

net.ipv4.tcp\_fin\_timeout = 30

net.ipv4.tcp\_keepalive\_time = 30

net.ipv4.ip\_local\_port\_range = 1024 65535

2）Nginx配置：/usr/local/nginx/conf/nginx.conf

worker\_processes 8;

error\_log logs/error.log;

pid logs/nginx.pid;

worker\_rlimit\_nofile 102400;

events {

worker\_connections 65535;

use epoll;

}

http {

include mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#access\_log logs/access.log main;

sendfile on;

tcp\_nopush on;

open\_file\_cache max=65535 inactive=20s;

open\_file\_cache\_min\_uses 1;

open\_file\_cache\_valid 30s;

tcp\_nodelay on;

#keepalive\_timeout 0;

keepalive\_timeout 65;

gzip on;

gzip\_min\_length 1k;

gzip\_buffers 4 16k;

gzip\_http\_version 1.0;

gzip\_comp\_level 2;

gzip\_types text/plain application/x-javascript text/css application/xml;

gzip\_vary on;

server {

listen 80;

server\_name localhost;

#charset koi8-r;

#access\_log logs/host.access.log main;

location ~.\*\.(jpg|png|jpeg)$

{

root /usr/local/apache-tomcat/webapps/ROOT;

expires 30d;

}

location ~.\*\.(css|js)$

{

root /usr/local/apache-tomcat/webapps/ROOT;

expires 1d;

}

location /

{

#root html;

#index index.html index.htm;

proxy\_pass http://localhost:8080;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

client\_max\_body\_size 500m;

client\_body\_buffer\_size 512k;

proxy\_connect\_timeout 180;

proxy\_send\_timeout 180;

proxy\_read\_timeout 90;

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

}

location /status

{

stub\_status on;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

upstream tomcats {

ip\_hash;

server 10.228.1.144:8080 weight=5 max\_fails=2 fail\_timeout=30s;

server 10.228.1.145:8080 weight=2 max\_fails=2 fail\_timeout=30s;

}

server {

listen 80;

server\_name test.css.org;

location ~.\*\.(jpg|png|jpeg)$

{

root /usr/local/apache-tomcat/webapps/ROOT;

expires 30d;

}

location ~.\*\.(css|js)$

{

root /usr/local/apache-tomcat/webapps/ROOT;

expires 1d;

}

location /

{

#root html;

#index index.html index.htm;

proxy\_pass http://tomcats;

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

client\_max\_body\_size 500m;

client\_body\_buffer\_size 512k;

proxy\_connect\_timeout 800;

proxy\_send\_timeout 800;

proxy\_read\_timeout 800;

proxy\_buffer\_size 4k;

proxy\_buffers 4 32k;

proxy\_busy\_buffers\_size 64k;

proxy\_temp\_file\_write\_size 64k;

}

}

3）Tomcat配置：/usr/local/apache-tomcat/conf/server.xml

<Connector port="8080" protocol="HTTP/1.1"

maxThreads="2000"

minSpareThreads="100"

maxSpareThreads="300"

acceptCount="1000"

enableLookups="false"

connectionTimeout="20000"

redirectPort="8443"

URIEncoding="UTF-8" />

<!-- A "Connector" using the shared thread pool-->

JVM路径：/usr/local/apache-tomcat/bin/catalina.sh

JAVA\_OPTS='-Xms1024m

-Xmx8000m

-XX:PermSize=256M

-XX:MaxNewSize=256m

-XX:MaxPermSize=256m'

4）Mysql: /etc/my.cnf

[client]

port= 3306

socket = /www/mysql/mysql.sock

[mysqld]

Port = 3306

datadir=/www/mysql

socket = /www/mysql/mysql.sock

skip-external-locking

key\_buffer\_size = 16M

max\_allowed\_packet = 1M

table\_open\_cache = 64

sort\_buffer\_size = 512K

net\_buffer\_length = 8K

read\_buffer\_size = 256K

read\_rnd\_buffer\_size = 512K

myisam\_sort\_buffer\_size = 8M

lower\_case\_table\_names=1

binlog-do-db=oec

binlog-do-db=oec\_test

binlog-ignore-db=mysql

binlog-ignore-db=information\_schema

binlog-ignore-db=performance\_schema

log-error = /var/log/mysql/mysql\_error.log

long\_query\_time = 5

max\_connections=65535

log-bin=/var/log/mysql/mysql\_bin.log

slow\_query\_log = 1

slow\_query\_log\_file = /var/log/mysql/mysql\_slow.log

binlog\_format=mixed

server-id = 1

innodb\_data\_home\_dir = /www/mysql

innodb\_data\_file\_path = ibdata1:10M:autoextend

innodb\_log\_group\_home\_dir = /www/mysql

innodb\_buffer\_pool\_size = 4000M

innodb\_additional\_mem\_pool\_size = 128M

innodb\_log\_file\_size = 256M

innodb\_log\_buffer\_size = 8M

innodb\_flush\_log\_at\_trx\_commit = 1

innodb\_lock\_wait\_timeout = 50

innodb\_use\_sys\_malloc = 1

[mysqldump]

quick

max\_allowed\_packet = 16M

[mysql]

no-auto-rehash

[myisamchk]

key\_buffer\_size = 20M

sort\_buffer\_size = 20M

read\_buffer = 2M

write\_buffer = 2M

[mysqlhotcopy]

interactive-timeout

**3. 硬件扩容策略**

当网站发展到一定阶段，随着用户量不断扩大，现有的网络资源和服务器资源不能满足用户需要的时候，就需要对平台进行服务器和网络的扩容。以下是两种平台扩容的方式：

### 3.1 增加服务器

对于web的并发处理有瓶颈时，新增的web服务器，把新增的web服务器填加到Web服务器集群中，以增加WEB的并发处理能力。

对于数据库有处理压力时，可以增加数据库服务器，增加数据库服务器加入数据库的集群中。

### 3.2 增加存储

对于存储容量不能满足业务需要时，可以考虑在磁盘柜中新增加硬盘，甚至考虑新增磁盘柜。

### 3.3 升级服务器

可以升级服务器的内存、硬盘，甚至考虑用新的性能更高的服务器来替换。

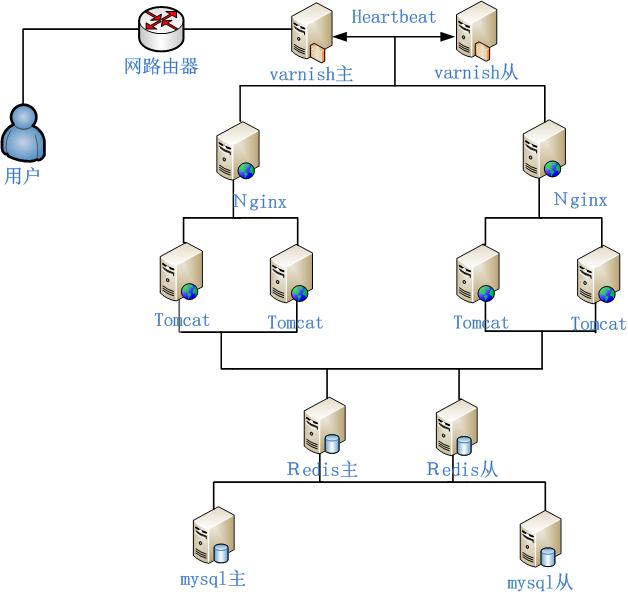
### 网络扩容

1）申请更大的网络带宽

2)引入CDN

3)升级内网交换机。

3.5 网络架构升级



##### varnish：用于反向代理将静态页图、图片缓存到内存中，用户访问静态数据时直接从内存中读取减少了nginx压力及I/o开销、实现负载均衡。

##### Hearbeat:用于心跳检测，当varnish主服务器宕机后备机马上接替工作保证业务正常运行。

##### Nginx:通过nginx实现对tomcat负载均衡提高tomcat并发数、事务成功率、处理静态页面实现动静分离。

##### Tomcat处理动态JAVA、耗CPU、内存，单台处理并发数为1000时CPU资源被耗尽，top显示已达到130左右，测试所得单台Tomcat并发数处理能力在600这内。而多台Tomcat的最大并发性低于单台Tomcat,但持续稳定服务器时间延长与Tomcat台数有关，2台Tomcat比单台要延长一倍。

##### Redis：主要用于缓存mysql数据，让程序直接去内存中取数据，如果没命中则在从mysql中取，这样减少IO开消，能做主从数据同步，不会因为宕机造成大理请求直接从mysqk取数据

##### Mysql:通过主从备份进行容灾，当主库宕机时自动切换到备库保证业务正常运行，因Redis做了内存缓存mysql数据所以没做mysql读写分离。