**Nginx实现高可用**

# nginx入门

## 什么是nginx？

nginx是一款高性能的http 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器。由俄罗斯的程序设计师Igor Sysoev所开发，官方测试nginx能够支支撑5万并发链接，并且cpu、内存等资源消耗却非常低，运行非常稳定，所以现在很多知名的公司都在使用nginx。

## 反向代理服务器

Nginx、lvs、F5（硬件）、haproxy

## 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主机。

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，不会因为某台服务器负载高宕机而某台服务器闲置的情况。



## Windows环境下安装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

### windows常用命令

nginx.exe -s stop –停止

## nginx优缺点

占内存小，可以实现高并发连接、处理响应快。

可以实现http服务器、虚拟主机、反向代理、负载均衡。

nginx配置简单

可以不暴露真实服务器IP地址

## nginx.conf 介绍

### nginx.conf文件的结构

nginx的配置由特定的标识符(指令符)分为多个不同的模块。   
指令符分为**简单指令**和**块指令**。

* 简单指令格式：[name parameters;]
* 块指令格式：和简单指令格式有一样的结构，但其结束标识符不是分号，而是大括号{},块指令内部可以包含simple directives 和block directives, 可以称块指令为上下文(e.g. events, http, server, location)

conf文件中，所有不属于块指令的简单指令都属于main上下文的，http块指令属于main上下文，server块指令http上下文。

### 配置静态访问

Web server很重要一部分工作就是提供静态页面的访问，例如images, html page。nginx可以通过不同的配置，根据request请求，从本地的目录提供不同的文件返回给客户端。   
打开安装目录下的nginx.conf文件，默认配置文件已经在http指令块中创建了一个空的server块，在nginx-1.8.0中的http块中已经创建了一个默认的server块。内容如下：

|  |
| --- |
| **server {**  **listen 80;**  **server\_name localhost;**  **location / {**  **root html;**  **index index.html index.htm;**  **}**  **error\_page 500 502 503 504 /50x.html;**  **location = /50x.html {**  **root html;**  **}**  **}** |

## nginx实现反向代理

### 什么是反向代理?

反向代理（Reverse Proxy）方式是指以[代理服务器](https://baike.baidu.com/item/%E4%BB%A3%E7%90%86%E6%9C%8D%E5%8A%A1%E5%99%A8)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

### Host文件新增

|  |
| --- |
| 127.0.0.1 8080.itmayiedu.com  127.0.0.1 8081.itmayiedu.com |

### nginx.conf 配置

配置信息:

|  |
| --- |
| server {  listen 80;  server\_name 8080.itmayiedu.com;  location / {  proxy\_pass http://127.0.0.1:8080;  index index.html index.htm;  }  }  server {  listen 80;  server\_name 8081.itmayiedu.com;  location / {  proxy\_pass http://127.0.0.1:8081;  index index.html index.htm;  }  } |

## nginx实现负载均衡

### 什么是负载均衡

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。



### 负载均衡策略

|  |
| --- |
| 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器，如果后端服务器down掉，能自动剔除。  upstream backserver {  server 192.168.0.14;  server 192.168.0.15;  }   **2、指定权重** 指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。  upstream backserver {  server 192.168.0.14 weight=10;  server 192.168.0.15 weight=10;  }   **3、IP绑定 ip\_hash** 每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。  upstream backserver {  ip\_hash;  server 192.168.0.14:88;  server 192.168.0.15:80;  }   **4、fair（第三方）** 按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。  upstream backserver {  server server1;  server server2;  fair;  }  **5、url\_hash（第三方）** 按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。  upstream backserver {  server squid1:3128;  server squid2:3128;  hash $request\_uri;  hash\_method crc32;  } |

### 配置代码

|  |
| --- |
| upstream backserver {  server 127.0.0.1:8080;  server 127.0.0.1:8081;  }  server {  listen 80;  server\_name www.itmayiedu.com;  location / {  proxy\_pass http://backserver;  index index.html index.htm;  }  } |

### 宕机轮训配置规则

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  location / {  proxy\_pass http://backserver;  index index.html index.htm;  proxy\_connect\_timeout 1;  proxy\_send\_timeout 1;  proxy\_read\_timeout 1;  }    } |

## nginx解决网站跨域问题

配置:

|  |
| --- |
| server {  listen 80;  server\_name www.itmayiedu.com;  location /A {  proxy\_pass http://a.a.com:81/A;  index index.html index.htm;  }  location /B {  proxy\_pass http://b.b.com:81/B;  index index.html index.htm;  }  } |

## nginx配置防盗链

|  |
| --- |
| location ~ .\*\.(jpg|jpeg|JPG|png|gif|icon)$ {  valid\_referers blocked http://www.itmayiedu.com www.itmayiedu.com;  if ($invalid\_referer) {  return 403;  }  } |

## nginx配置DDOS

### 限制请求次数

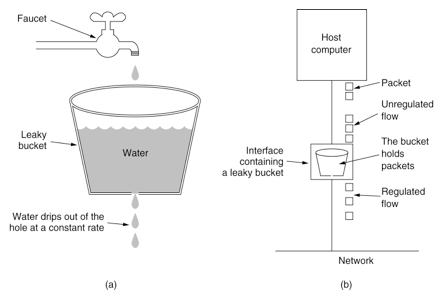
设置Nginx、Nginx Plus的连接请求在一个真实用户请求的合理范围内。比如，如果你觉得一个正常用户每两秒可以请求一次登录页面，你就可以设置Nginx每两秒钟接收一个客户端IP的请求（大约等同于每分钟个请求）。

|  |
| --- |
| limit\_req\_zone $binary\_remote\_addr zone=one:10m rate=2r/s;  server {  ...  location /login.html {  limit\_req zone=one;  ...  }  } |

`limit\_req\_zone`命令设置了一个叫one的共享内存区来存储请求状态的特定键值，在上面的例子中是客户端IP($binary\_remote\_addr)。location块中的`limit\_req`通过引用one共享内存区来实现限制访问/login.html的目的。

### 限制请求速度

设置Nginx、Nginx Plus的连接数在一个真实用户请求的合理范围内。比如，你可以设置每个客户端IP连接/store不可以超过10个。

漏桶算法可以很好地限制容量池的大小，从而防止流量暴增。如果针对uri+ip作为监测的key，就可以实现定向的设定指定ip对指定uri容量大小，超出的请求做队列处理（队列处理要引入消息机制）或者丢弃处理。这也是v2ex对流量拦截的算法，针对uri+ip做流量监测。   


# linux操作nginx

Nginx是一款轻量级的网页服务器、反向代理服务器。相较于Apache、lighttpd具有占有内存少，稳定性高等优势。**它最常的用途是提供反向代理服务。**

## linux安装

在Centos下，yum源不提供nginx的安装，可以通过切换yum源的方法获取安装。也可以通过直接下载安装包的方法，以下命令均需root权限执行：

首先安装必要的库（nginx 中gzip模块需要 zlib 库，rewrite模块需要 pcre 库，ssl 功能需要openssl库）。选定/usr/local为安装目录，以下具体版本号根据实际改变。

### 安装PCRE库

|  |
| --- |
| $ cd /usr/local/  $ wget ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/pcre-8.36.tar.gz  $ tar -zxvf pcre-8.36.tar.gz  $ cd pcre-8.36  $ ./configure  $ make  $ make install |

./configure报错

configure: error: You need a C++ compiler for C++ support.

解决办法

yum install -y gcc gcc-c++

### 安装zlib库

|  |
| --- |
| $ cd /usr/local/  $ wget http://zlib.net/zlib-1.2.8.tar.gz  $ tar -zxvf zlib-1.2.8.tar.gz  $ cd zlib-1.2.8  $ ./configure  $ make  $ make install |

### 安装ssl

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

### 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 --prefix=/usr/local/nginx  $ make  $ make install |

安装常见错误:

Nginx启动提示找不到libpcre.so.1解决方法

**如果是32位系统**

[root@lee ~]#  ln -s /usr/local/lib/libpcre.so.1 /lib

**如果是64位系统**

[root@lee ~]#  ln -s /usr/local/lib/libpcre.so.1 /lib64

**然后在启动nginx就OK了**

[root@lee ~]# /usr/local/webserver/nginx/sbin/nginx

## 启动nginx

$ /usr/local/nginx/sbin/nginx

检查是否启动成功：

打开浏览器访问此机器的 IP，如果浏览器出现 Welcome to nginx! 则表示 Nginx 已经安装并运行成功。

### 常用命令

重启：  
$ /usr/local/nginx/sbin/nginx 启动命令

重启：  
$ /usr/local/nginx/sbin/nginx –s reload

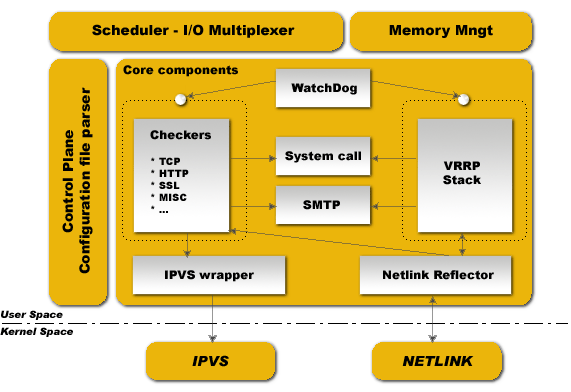
停止：  
$ /usr/local/nginx/sbin/nginx –s stop

测试配置文件是否正常：  
$ /usr/local/nginx/sbin/nginx –t

强制关闭：  
$ pkill nginx

## 启动Nginx + Keepalived高可用

## 什么是Keepalived

[Keepalived](http://www.keepalived.org/)是一个免费开源的，用C编写的类似于layer3, 4 & 7交换机制软件，具备我们平时说的第3层、第4层和第7层交换机的功能。主要提供**loadbalancing**（负载均衡）和 **high-availability**（高可用）功能，负载均衡实现需要依赖Linux的虚拟服务内核模块（ipvs），而高可用是通过VRRP协议实现多台机器之间的故障转移服务。   
   
上图是Keepalived的功能体系结构，大致分两层：用户空间（user space）和内核空间（kernel space）。   
**内核空间**：主要包括IPVS（IP虚拟服务器，用于实现网络服务的负载均衡）和NETLINK（提供高级路由及其他相关的网络功能）两个部份。   
**用户空间**：

* WatchDog：负载监控checkers和VRRP进程的状况
* VRRP Stack：负载负载均衡器之间的失败切换FailOver，如果只用一个负载均稀器，则VRRP不是必须的。
* Checkers：负责真实服务器的健康检查healthchecking，是keepalived最主要的功能。换言之，可以没有VRRP Stack，但健康检查healthchecking是一定要有的。
* IPVS wrapper：用户发送设定的规则到内核ipvs代码
* Netlink Reflector：用来设定vrrp的vip地址等。

Keepalived的所有功能是配置keepalived.conf文件来实现的。

### 安装keepalived

下载keepalived地址：http://www.keepalived.org/download.html

解压安装：

tar -zxvf keepalived-1.2.18.tar.gz -C /usr/local/

yum install -y openssl openssl-devel（需要安装一个软件包）

cd keepalived-1.2.18/ && ./configure --prefix=/usr/local/keepalived

make && make install

### keepalived安装成Linux系统服务

将keepalived安装成Linux系统服务，因为没有使用keepalived的默认安装路径（默认路径：/usr/local）,安装完成之后，需要做一些修改工作：

首先创建文件夹，将keepalived配置文件进行复制：

mkdir /etc/keepalived

cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/

然后复制keepalived脚本文件：

cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/

cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/

ln -s /usr/local/sbin/keepalived /usr/sbin/

ln -s /usr/local/keepalived/sbin/keepalived /sbin/

可以设置开机启动：chkconfig keepalived on，到此我们安装完毕!

### keepalived 常用命令

service keepalived start

service keepalived stop

### 配置nginx主备自动重启

第三步：对配置文件进行修改：vim /etc/keepalived/keepalived.conf

keepalived.conf配置文件说明：

（一）Master

! Configuration File for keepalived

global\_defs {

router\_id bhz005 ##标识节点的字符串，通常为hostname

}

## keepalived 会定时执行脚本并且对脚本的执行结果进行分析，动态调整vrrp\_instance的优先级。这里的权重weight 是与下面的优先级priority有关，如果执行了一次检查脚本成功，则权重会-20，也就是由100 - 20 变成了80，Master 的优先级为80 就低于了Backup的优先级90，那么会进行自动的主备切换。

如果脚本执行结果为0并且weight配置的值大于0，则优先级会相应增加。

如果脚本执行结果不为0 并且weight配置的值小于0，则优先级会相应减少。

vrrp\_script chk\_nginx {

script "/etc/keepalived/nginx\_check.sh" ##执行脚本位置

interval 2 ##检测时间间隔

weight -20 ## 如果条件成立则权重减20（-20）

}

## 定义虚拟路由 VI\_1为自定义标识。

vrrp\_instance VI\_1 {

state MASTER ## 主节点为MASTER，备份节点为BACKUP

## 绑定虚拟IP的网络接口（网卡），与本机IP地址所在的网络接口相同（我这里是eth6）

interface eth6

virtual\_router\_id 172 ## 虚拟路由ID号

mcast\_src\_ip 192.168.1.172 ## 本机ip地址

priority 100 ##优先级配置（0-254的值）

Nopreempt ##

advert\_int 1 ## 组播信息发送间隔，俩个节点必须配置一致，默认1s

authentication {

auth\_type PASS

auth\_pass bhz ## 真实生产环境下对密码进行匹配

}

track\_script {

chk\_nginx

}

virtual\_ipaddress {

192.168.1.170 ## 虚拟ip(vip)，可以指定多个

}

}

（二）Backup

! Configuration File for keepalived

global\_defs {

router\_id bhz006

}

vrrp\_script chk\_nginx {

script "/etc/keepalived/nginx\_check.sh"

interval 2

weight -20

}

vrrp\_instance VI\_1 {

state BACKUP

interface eth7

virtual\_router\_id 173

mcast\_src\_ip 192.168.1.173

priority 90 ##优先级配置

advert\_int 1

authentication {

auth\_type PASS

auth\_pass bhz

}

track\_script {

chk\_nginx

}

virtual\_ipaddress {

192.168.1.170

}

}

（三）nginx\_check.sh 脚本：

#!/bin/bash

A=`ps -C nginx –no-header |wc -l`

if [ $A -eq 0 ];then

/usr/local/nginx/sbin/nginx

sleep 2

if [ `ps -C nginx --no-header |wc -l` -eq 0 ];then

killall keepalived

fi

fi

1. 我们需要把master的keepalived配置文件 copy到master机器（172）的 /etc/keepalived/ 文件夹下，在把backup的keepalived配置文件copy到backup机器（173）的 /etc/keepalived/ 文件夹下，最后把nginx\_check.sh脚本分别copy到两台机器的 /etc/keepalived/文件夹下。
2. nginx\_check.sh脚本授权。赋予可执行权限：chmod +x /etc/keepalived/nginx\_check.sh
3. 启动2台机器的nginx之后。我们启动两台机器的keepalived

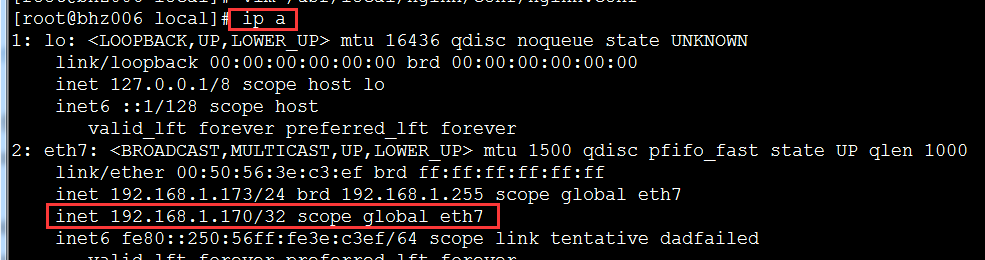
/usr/local/nginx/sbin/nginx

service keepalived start

ps -ef | grep nginx

ps -ef | grep keepalived

可以进行测试，首先看一下俩台机器的ip a 命令下 都会出现一个虚拟ip，我们可以停掉 一个机器的keepalived，然后测试，命令：service keepalived stop。结果发现当前停掉的机器已经不可用，keepalived会自动切换到另一台机器上。



1. 我们可以测试在nginx出现问题的情况下，实现切换，这个时候我们只需要把nginx的配置文件进行修改，让其变得不可用，然后强杀掉nginx进程即可，发现也会实现自动切换服务器节点。

## nginx配置负载均衡

创建一个springboot项目 实现负载均衡

# 集群情况下Session共享解决方案

## 集群情况下session会产生什么原因?

## 什么是会话机制

## Session存放在哪里

## Session共享解决方案

### nginx或者haproxy做的负载均衡)

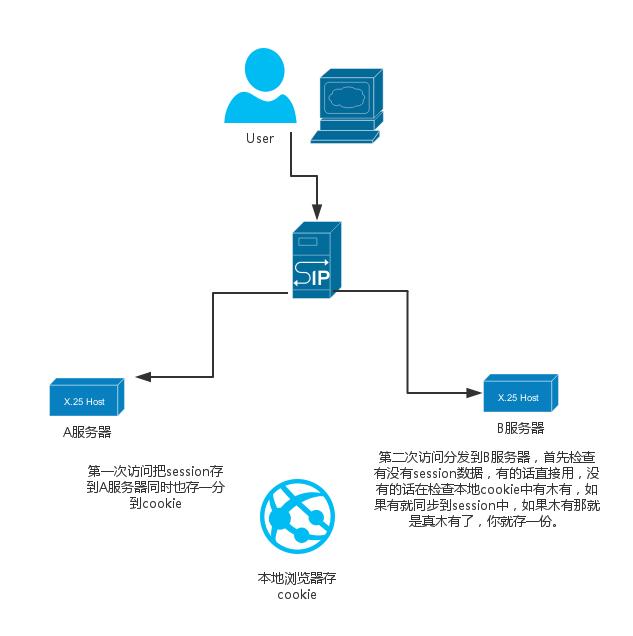
用Nginx 做的负载均衡可以添加ip\_hash这个配置，

用haproxy做的负载均衡可以用 balance source这个配置。

从而使同一个ip的请求发到同一台服务器。

### 利用数据库同步session

### 利用cookie同步session数据原理图如下



缺点:安全性差、http请求都需要带参数增加了带宽消耗

### 使用Session集群存放Redis

创建一个springboot项目

#### 引入maven依赖

|  |
| --- |
| **<!--spring boot 与redis应用基本环境配置 -->**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-redis</artifactId>**  **</dependency>**  **<!--spring session 与redis应用基本环境配置,需要开启redis后才可以使用，不然启动Spring boot会报错 -->**  **<dependency>**  **<groupId>org.springframework.session</groupId>**  **<artifactId>spring-session-data-redis</artifactId>**  **</dependency>** |

#### 创建SessionConfig

|  |
| --- |
| **import org.springframework.beans.factory.annotation.Value;**  **import org.springframework.context.annotation.Bean;**  **import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;**  **import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;**  **//这个类用配置redis服务器的连接**  **//maxInactiveIntervalInSeconds为SpringSession的过期时间（单位：秒）**  **@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 1800)**  **public class SessionConfig {**  **// 冒号后的值为没有配置文件时，制动装载的默认值**  **@Value("${redis.hostname:localhost}")**  **String HostName;**  **@Value("${redis.port:6379}")**  **int Port;**  **@Bean**  **public JedisConnectionFactory connectionFactory() {**  **JedisConnectionFactory connection = new JedisConnectionFactory();**  **connection.setPort(Port);**  **connection.setHostName(HostName);**  **return connection;**  **}**  **}** |

#### 初始化Session

|  |
| --- |
| **//初始化Session配置**  **public class SessionInitializer extends AbstractHttpSessionApplicationInitializer{**  **public SessionInitializer() {**  **super(SessionConfig.class);**  **}**  **}** |

#### 控制器层代码

|  |
| --- |
| import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpSession;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.boot.SpringApplication;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class SessionController {  @Value("${server.port}")  private String PORT;  public static void main(String[] args) {  SpringApplication.run(SessionController.class, args);  }  @RequestMapping("/index")  public String index() {  return "index:" + PORT;  }  /\*\*  \*  \* @methodDesc: 功能描述:(往session存放值)  \* @author: 余胜军  \* @param: @param  \* httpSession  \* @param: @param  \* sessionKey  \* @param: @param  \* sessionValue  \* @param: @return  \* @createTime:2017年10月8日 下午3:55:26  \* @returnType:@param httpSession  \* @returnType:@param sessionKey  \* @returnType:@param sessionValue  \* @returnType:@return String  \* @copyright:上海每特教育科技有限公司  \* @QQ:644064779  \*/  @RequestMapping("/setSession")  public String setSession(HttpServletRequest request, String sessionKey, String sessionValue) {  HttpSession session = request.getSession(true);  session.setAttribute(sessionKey, sessionValue);  return "success,port:" + PORT;  }  /\*\*  \*  \* @methodDesc: 功能描述:(从Session获取值)  \* @author: 余胜军  \* @param: @param  \* httpSession  \* @param: @param  \* sessionKey  \* @param: @return  \* @createTime:2017年10月8日 下午3:55:47  \* @returnType:@param httpSession  \* @returnType:@param sessionKey  \* @returnType:@return String  \* @copyright:上海每特教育科技有限公司  \* @QQ:644064779  \*/  @RequestMapping("/getSession")  public String getSession(HttpServletRequest request, String sessionKey) {  HttpSession session =null;  try {  session = request.getSession(false);  } catch (Exception e) {  e.printStackTrace();  }  String value=null;  if(session!=null){  value = (String) session.getAttribute(sessionKey);  }  return "sessionValue:" + value + ",port:" + PORT;  }  } |

# 高并发解决方案

业务数据库  -》 数据水平分割(分区分表分库)、读写分离

业务应用 -》 逻辑代码优化(算法优化)、公共数据缓存

应用服务器 -》 反向静态代理、配置优化、负载均衡(apache分发，多tomcat实例)

系统环境 -》 JVM调优

页面优化 -》 减少页面连接数、页面尺寸瘦身

1、动态资源和静态资源分离；

2、CDN；

3、负载均衡；

4、分布式缓存；

5、数据库读写分离或数据切分（垂直或水平）；

6、服务分布式部署。