1. 负载均衡目的：

将前端超高并发访问转发至后端多台服务器进行处理，解决单个节点压力过大，造成Web服务响应过慢，严重的情况下导致服务瘫痪，无法正常提供服务的问题。

2. 工作原理：

负载均衡分为四层负载均衡和七层负载均衡。

四层负载均衡是工作在七层协议的第四层-传输层，主要工作是**转发**。

它在接收到客户端的流量以后通过修改数据包的地址信息（目标地址和端口和源地址）将流量转发到应用服务器。

七层负载均衡是工作在七层协议的第七层-应用层，主要工作是**代理**。

它首先会与客户端建立一条完整的连接并将应用层的请求流量解析出来，再按照调度算法选择一个应用服务器，并与应用服务器建立另外一条连接将请求发送过去。

OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	HTTP、TFTP, FTP, NFS, WAIS、SMTP
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	数据链路层	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

3. 配置七层均衡：

```
前端服务器：192.168.1.6

后端服务器1：192.168.1.5
后端服务器2：192.168.1.7

这里后端服务器也可以通过配置虚拟主机实现。
```

前端服务器主要配置upstream和proxy_pass：

upstream　主要是配置均衡池和调度方法。

proxy_pass　主要是配置代理服务器ip或服务器组的名字。

proxy_set_header　主要是配置转发给后端服务器的Host和前端客户端真实ip。

3.1 配置前端nginx：

```
# 在http指令块下配置upstream指令块

    upstream web {
        server 192.168.1.5;
        server 192.168.1.7;
    }

 # 在location指令块配置proxy_pass
    server {
        listen       80;
        server_name  localhost;
        location /  {
            proxy_pass http://web;
            proxy_next_upstream  error http_404 http_502;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }

# proxy_next_upstream  error http_404 http_502;
通过这个指令，可以处理当后端服务返回404等报错时，
直接将请求转发给其他服务器，而不是把报错信息返回客户端。
# proxy_set_header Host $host;
通过这个指令，把客户端请求的host，转发给后端。
```

```
# proxy_set_header X-Real-IP $remote_addr
```
通过这个指令，把客户端的IP转发给后端服务器，在后端服务器的日志格式中，添加$http_x_real_ip即可获取原始客户端的IP了。

3.2 配置后端nginx：

```
yum install nginx -y
echo "this is 1.5 page" > /usr/share/nginx/html/index.html
echo "this is 1.7 page" > /usr/share/nginx/html/index.html
```

3.3 均衡方式：

3.3.1 轮询:

```
upstream web {
    server 192.168.1.5;
    server 192.168.1.7;
    }
```

访问前端IP：

```
[root@192 ~]# while true;do curl 192.168.1.6;sleep 2;done
this is 1.5  page
                        this is 1.7  page
this is 1.5  page
                        this is 1.7  page

#可以看到后端服务器，非常平均的处理请求。
```

3.3.2 轮询加权重:

```
upstream web {
        server 192.168.1.5 weight=3;
        server 192.168.1.7 weight=1;
        }
```

访问前端IP：

```
[root@192 ~]# while true;do curl 192.168.1.6;sleep 1;done
this is 1.5  page
this is 1.5  page
this is 1.5  page
                      this is 1.7  page
this is 1.5  page
this is 1.5  page
this is 1.5  page
                      this is 1.7  page

#后端服务，根据权重比例处理请求，适用于服务器性能不均的环境。
```

3.3.3 最大错误连接次数:

错误的连接由proxy_next_upstream， fastcgi_next_upstream等指令决定，且默认情况下，后端某台服务器**出现故障了**，nginx会自动将请求再次转发给其他正常的服务器（因为默认 proxy_next_upstream error timeout）。所以即使我们没有配这个参数，nginx也可以帮我们处理error和timeout的响应，但是没法处理404等报错。

为了看清楚本质，**可以先将proxy_next_upstream设置为off**，也就是不将失败的请求转发给其他正常服务器，这样我们可以看到请求失败的结果。

```
upstream web {
        server 192.168.1.5 weight=1  max_fails=3
fail_timeout=9s;
        #先将1.5这台nginx关了。
        server 192.168.1.7 weight=1;
        }


server {
        listen       80;
        server_name  localhost;
        location / {
            proxy_pass http://web;
            #proxy_next_upstream  error http_404 http_502;
            proxy_next_upstream off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
```

```
        }
}
# 在这里，我们将超时时间设置为9s，最多尝试3次，
这里要注意，尝试3次，依然遵循轮询的规则，并不是一个请求，连接3次，
而是轮询三次，有3次处理请求的机会。
```

访问前端IP：

```
[root@192 ~]# while true;do curl -I 192.168.1.6
2>/dev/null|grep HTTP/1.1 ;sleep 3;done
HTTP/1.1 502 Bad Gateway
HTTP/1.1 200 OK
HTTP/1.1 502 Bad Gateway
HTTP/1.1 200 OK
HTTP/1.1 502 Bad Gateway
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 502 Bad Gateway

# 我们设置的超时时间为9s，我们是每3s请求一次。
我们可以看到后端一台服务器挂了后，请求没有直接转发给正常的服务器，
而是直接返回了502。尝试三次后，等待9s，才开始再次尝试（最后一个
502）。
```

把proxy_next_upstream开启，再来访问看结果：

```
upstream web {
        server 192.168.1.5 weight=1  max_fails=3
fail_timeout=9s;
        #先将1.5这台nginx关了。
        server 192.168.1.7 weight=1;
        }

server {
        listen       80;
        server_name  localhost;
        location / {
            proxy_pass http://web;
```

```
            proxy_next_upstream  error http_404 http_502;
            #proxy_next_upstream off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
}
```

访问前端IP：

```
[root@192 ~]# while true;do curl -I 192.168.1.6
2>/dev/null|grep HTTP/1.1 ;sleep 3;done
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK

#可以看到现在没有502报错，请求都处理了。因为错误的响应码被
proxy_next_upstream 获取，这次请求被转发给下一个正常的服务器了。
```

3.3.4 ip_hash：

通过客户端ip进行hash，再通过hash值选择后端server 。

```
upstream web {
        ip_hash;
        server 192.168.1.5 weight=1  max_fails=3
fail_timeout=9s;
        server 192.168.1.7 weight=1;
        }

server {
        listen       80;
        server_name  localhost;
        location / {
            proxy_pass http://web;
            proxy_next_upstream  error http_404 http_502;
            #proxy_next_upstream off;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
```

```
}
```

访问前端IP：

```
[root@192 ~]# while true;do curl 192.168.1.6;sleep 2;done
this is 1.5  page
this is 1.5  page
this is 1.5  page
^C
[root@192 ~]# curl 192.168.1.7
                        this is 1.7  page

#可以看到1.7的服务是正常的，但是却不转发给1.7了，请求固定在了1.5的服务器上。
```

在使用负载均衡的时候会遇到会话保持的问题,常用的方法有:

a、ip_hash，根据客户端的IP，将请求分配到不同的服务器上;

b、cookie，服务器给客户端下发一个cookie，具有特定cookie的请求会分配给它的发布者。

3.3.5 url_hash:

通过请求url进行hash，再通过hash值选择后端server

```
upstream nginx_web {
                hash $request_uri consistent;
                server 192.168.75.128;
                server 192.168.75.135;
        }
```

3.3.5 根据响应时间均衡;

fair算法会根据后端节点服务器的响应时间来分配请求，时间短的优先分配

```
# 下载模块：
wget https://github.com/gnosek/nginx-upstream-
fair/archive/master.zip
# 解压：
```

```
unzip  master.zip
# 修改源码bug：
sed -i 's/default_port/no_port/g'
ngx_http_upstream_fair_module.c
# 预编译：
./configure --prefix=/usr/local/nginx --add-
module=../echo-nginx-module --with-http_stub_status_module
--add-module=../nginx-upstream-fair-master
# 编译/安装：
make && make install

# 配置：
upstream web {
        fair;
        server 192.168.1.5 weight=1  max_fails=3
fail_timeout=9s;
        server 192.168.1.7 weight=1;
        }
```

3.3.6 备用服务器：

```
upstream web {
        server 192.168.1.5 weight=1  max_fails=3
fail_timeout=9s;
        server 192.168.1.7 weight=1 backup;
        }

# 1.7的服务器做备用服务器，只有在1.5得服务器不能提供服务时，才会自动顶上,否则，默认是不提供服务的。
```

4. 配置四层均衡：

```
前端服务器：192.168.75.130

后端服务器1：192.168.75.128
后端服务器2：192.168.75.135
```

前端服务器主要配置stream和upstream，注意该模块需要在预编译时指定，没有被默认编译进nginx。

4.1 配置前端服务器：

```
# 预编译：
./configure --prefix=/usr/local/nginx --add-
module=../echo-nginx-module --with-http_stub_status_module
 --with-stream
# 编译/安装：
make && make install

# 在main全局配置stream：
events {
    worker_connections  1024;
}

stream {
        upstream web {
                # 必须要指定ip加port
                server 192.168.75.128:80;
                server 192.168.75.135:80;
        }

        server {
                listen 80;
                # 连接上游服务器超时间，超过则选择另外一个服务器
                proxy_connect_timeout 3s;
                # tcp连接闲置时间，超过则关闭
                proxy_timeout 10s;
                proxy_pass web;
        }
        log_format  proxy '$remote_addr $remote_port
$protocol $status  [$time_iso8601] '
                          '"$upstream_addr"
"$upstream_bytes_sent" "$upstream_connect_time"' ;
       access_log /usr/local/nginx/logs/proxy.log proxy;
}
```

4.2 配置后端测试页面：

```
echo "this is 128 page" > /usr/local/nginx/html/index.html
echo "this is 135 page" > /usr/local/nginx/html/index.html
```

4.3 访问前端服务器：

```
# 在后端135上访问130：
[root@node5 ~]# curl 192.168.75.130/index.html
this is 128 page
# 查看后端128日志：
[root@node2 ~]# tailf /usr/local/nginx/logs/access.log
192.168.75.130 - - [17/Dec/2019:11:07:56 +0800] "GET
/index.html HTTP/1.1" 200 17 "-" "curl/7.29.0" "-"
# 查看前端130代理日志：
[root@node3 nginx-1.16.0]# tailf
/usr/local/nginx/logs/proxy.log
192.168.75.135 57704 TCP 200  [2019-12-17T11:07:56+08:00]
"192.168.75.128:80" "88" "0.001"
```

4.4. 端口转发：

```
# 前端130上配置如下：
stream {
        upstream web {
                server 192.168.75.128:22;
        }
        server {
                listen 2222;
                proxy_connect_timeout 3s;
                proxy_timeout 10s;
                proxy_pass web;
                #proxy_set_header X-Real-IP $remote_addr;
        }
        log_format  proxy '$remote_addr $remote_port
$protocol $status  [$time_iso8601] '
                          '"$upstream_addr"
"$upstream_bytes_sent" "$upstream_connect_time"' ;
       access_log /usr/local/nginx/logs/proxy.log proxy;
}

# 在另外一台服务器ssh连接：
[root@node5 ~]# ssh 192.168.75.130 -p 2222
root@192.168.75.130's password:
Last login: Wed Dec 18 15:50:37 2019 from 192.168.75.130
[root@node2 ~]#
```

```
[root@node2 ~]#
# 可以看到，已经登录到了128服务器上。
[root@node2 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP qlen 1000
    link/ether 00:50:56:2a:4e:7f brd ff:ff:ff:ff:ff:ff
    inet 192.168.75.128/24 brd 192.168.75.255 scope global
ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::52eb:5720:d625:841/64 scope link
       valid_lft forever preferred_lft forever
```

5. Mysql 数据库简介：

MySQL 是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。

关系数据库管理系统(Relational Database Management System，RDBMS)，是将数据组织为相关的行和列的系统，而管理关系数据库的计算机软件就是关系数据库管理系统 。

数据库一般分为以下两种：

- 关系型数据库；
- 非关系型数据库。

```
常用的关系型数据库软件有：MYSQL、Mariadb、Oracle、SQL Server、
PostgreSQL、DB2等
常用的非关系型数据库软件有：Redis、memcached。
```

MySQL现阶段有免费版本和付费版本，公司里面使用MySQL 多数都是使用免费版本。

Mariadb 数据库是MySQL 数据库原来的团队，后续独立出来进行开发的完全开源版本。

5.1 mysql引擎：

MySQL 引擎有很多，企业里面主流的myisam、innodb 两种。
MyISAM：主要强调的是性能，其执行数度比InnoDB类型更快，但不提供事务支持，不支持外键，如果执行大量的SELECT(查询)操作，MyISAM是更好的选择，支持表锁。myisam引擎查询性能高。

InnoDB：提供事务支持事务、外键、行级锁等高级数据库功能，可执行大量的INSERT或UPDATE，InnoDB引擎写性能高。

6. Mysql数据库安装：

MySQL安装方式有两种，一种是yum/rpm安装，另外一种是tar源码安装。

6.1 yum安装：

Yum 安装方法很简单，执行命令如下即可:

```
Centos6: yum install -y mysql-server  mysql-devel  mysql
Centos7:yum install -y  mariadb  mariadb-devel  mariadb-
server
```

```
# 安装完成之后可以查询一下软件是否安装成功：
[root@localhost ~]# rpm -qa|grep mariadb
mariadb-libs-5.5.60-1.el7_5.x86_64
mariadb-server-5.5.60-1.el7_5.x86_64
mariadb-5.5.60-1.el7_5.x86_64
mariadb-devel-5.5.60-1.el7_5.x86_64
Mariadb 和 mysql 数据库软件命令和配置都是类似的。
```

6.2 源码安装5.5版本：

```
cd /usr/src
wget http://mirrors.163.com/mysql/Downloads/MySQL-
5.5/mysql-5.5.60.tar.gz
tar xf mysql-5.5.60.tar.gz
cd mysql-5.5.60/
```

```
yum install gcc ncurses-devel libaio bison gcc-c++  git
cmake  ncurses-devel ncurses -y

cmake  .  -DCMAKE_INSTALL_PREFIX=/usr/local/mysql55/ \
-DMYSQL_UNIX_ADDR=/tmp/mysql.sock \
-DMYSQL_DATADIR=/data/mysql \
-DSYSCONFDIR=/usr/local/mysql55/ \
-DMYSQL_USER=mysql \
-DMYSQL_TCP_PORT=3306 \
-DWITH_XTRADB_STORAGE_ENGINE=1 \
-DWITH_INNOBASE_STORAGE_ENGINE=1 \
-DWITH_PARTITION_STORAGE_ENGINE=1 \
-DWITH_BLACKHOLE_STORAGE_ENGINE=1 \
-DWITH_MYISAM_STORAGE_ENGINE=1 \
-DWITH_READLINE=1 \
-DENABLED_LOCAL_INFILE=1 \
-DWITH_EXTRA_CHARSETS=1 \
-DDEFAULT_CHARSET=utf8 \
-DDEFAULT_COLLATION=utf8_general_ci \
-DEXTRA_CHARSETS=all \
-DWITH_BIG_TABLES=1 \
-DWITH_DEBUG=0

make  && make install

cp support-files/my-large.cnf /usr/local/mysql55/my.cnf
cp support-files/mysql.server /etc/init.d/mysqld
chmod +x  /etc/init.d/mysqld
mkdir -p /data/mysql
useradd -s /sbin/nologin mysql
chown -R mysql. /data/mysql
/usr/local/mysql55/scripts/mysql_install_db  --user=mysql
--datadir=/data/mysql --basedir=/usr/local/mysql55
/etc/init.d/mysqld start

加入service服务:
chkconfig --add mysqld
chkconfig --level 35 mysqld on
service mysqld restart
```

6.2.1 常见报错:

- 客户端连接报错：

```
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/lib/mysql/mysql.sock' (2)
# 分析原因：
找不到套接字，怎么找？
[root@node3 ~]# ps -ef |grep mysqld
root      5165     1  0 21:22 pts/0    00:00:00 /bin/sh
/usr/local/mysql55/bin/mysqld_safe --datadir=/data/mysql --
pid-file=/data/mysql/node3.pid
mysql     5536  5165  0 21:22 pts/0    00:00:01
/usr/local/mysql55/bin/mysqld --basedir=/usr/local/mysql55
--datadir=/data/mysql --plugin-
dir=/usr/local/mysql55/lib/plugin --user=mysql --log-
error=/var/log/mariadb/mariadb.log --pid-
file=/data/mysql/node3.pid --socket=/tmp/mysql.sock --
port=3306
解决方案一：
[root@node3 ~]# mysql -S /tmp/mysql.sock
解决方案二：
[root@node3 ~]# ln -s /tmp/mysql.sock
/var/lib/mysql/mysql.sock
解决方案三？
```