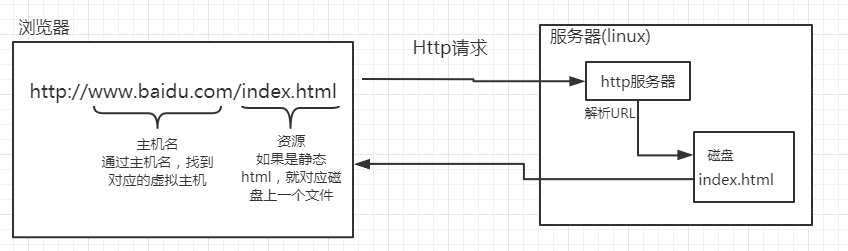
**Tengine和Nginx**

**Nginx是一个高性能的HTTP和反向代理服务器，也是一个IMAP/POP3/SMTP代理服务器。**

# 一、Web服务器

http服务器：实现了http协议的服务器。如果客户端（如浏览器）也实现了http协议，他就可以向http服务器发送请求。



Nginx（能解析到URL说明对数据包解析的层数比较深，7层网络）

【LVS：4层网络，Socket包拆到第4层，看不到URL，只能看到IP地址】

http服务器接收到用户请求后，URL经过主机名、资源 两层解析，能够定位到服务器磁盘上的某一个文件

文件返回给浏览器，浏览器拿到后解析展示

其他http服务器：Apache httpd，IIS(windows)

Tengine是Nginx的增强，在nginx基础上做的功能模块的增强。

Nginx本质上就一个功能，就是读文件

# 二、Nginx和apache的优缺点

## Nginx相对于Apache的优点：

1. 轻量级，同样起web服务，比Apache占用更少的内存及资源
2. 抗并发，nginx处理请求是异步非阻塞的，而Apache是阻塞型的，在高并发下nginx能保持低资源低消耗，高性能。
3. 高度模块化的设计，编写模块相对简单
4. 社区活跃，各种高性能模块出品迅速

## Apache相对于nginx的优点：

1. Rewrite，比nginx的rewrite强大
2. 模块超多，基本想到的都可以找到
3. 少bug，nginx的bug相对较多

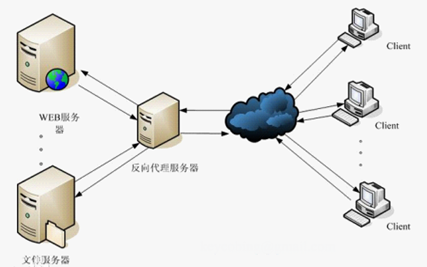
Nginx配置简介，Apache配置复杂

最核心的区别在于Apache是同步多进程模型，一个连接对应一个进程；

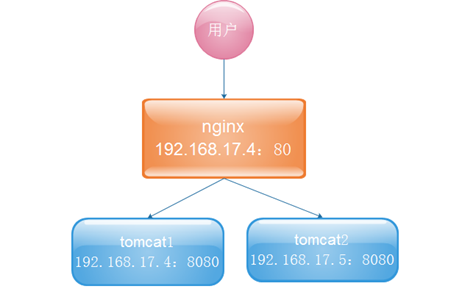
Nginx是异步的，多个连接（万级别）可以对应一个进程【一个进程fork出来一些子线程】

# 反向 （正向）代理 服务器

|  |
| --- |
| 通常的代理服务器，只用于代理内部网络对Internet的连接请求，客户机必须指定代理服务器,并将本来要直接发送到Web服务器上的http请求发送到代理服务器中由代理服务器向Internet上的web服务器发起请求，最终达到客户机上网的目的。  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 |

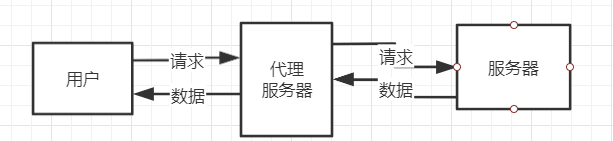


#### 经典的反向代理结构



正向、反向，占的角度不同。

【正向代理】站在用户角度，我想去上网，我想去看外国的网站，直接访问不了，代理服务器可以。



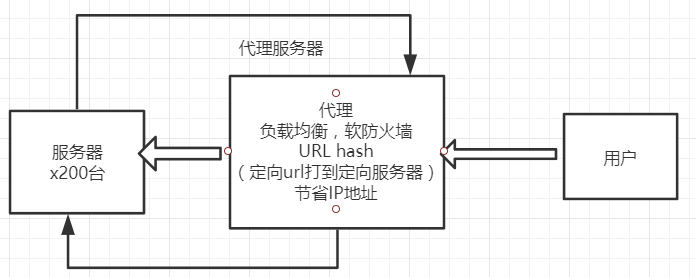
比如你在大型企业里上网，必须配置代理服务器，可以过滤上网，上班时间不允许看优酷，淘宝等。

有些路由网关，路由器支持编程，本质上就是一个代理服务器，可以监控用户的一些请求。

这是站在【用户的角度】来说，用户上网可以监控用户的请求。

反过来说，

【站在服务器】的角度，通过一个代理服务器得到用户请求的信息



代理服务器可以做负载均衡、

URL hash（使不同的url打到不同的后台服务器）

更节省IP（后台有200台服务器，和搭理服务器同处于一个网段，用户请求打到代理服务器，代理服务器转发给不同的后台服务器）

\*\*\*这里有一个问题是，用户将请求发给代理服务器，代理服务器将请求转发给后台服务器，后台服务器又将数据返回给代理服务器，代理服务器将结果返回给用户。

问题1：代理服务器负载高。所有请求都打到了代理这一台机器

问题2：代理服务器IO瓶颈。假如用户直接请求后台服务器，用户可以和后台服务器建立 点对点的连接。

现在只有用户---代理服务器 这一个点对点连接。 用户 《======》代理服务器 《======》后台服务器 ，所有后台服务器（比如200台）的数据都得通过代理服务器，再返回给用户。所以在IO密集型情况下，代理服务器也要做负载均衡

（在Nginx之上再加一层 LVS ，LVS用户请求到了nginx后转发给后台服务器，后台服务器返回的数据不经过nginx ，直接返回给用户，请求的信息都是些http请求参数信息，相对较小，除了你是一些文件上传的话，还是比较大的，一般系统架构的时候将上传文件的功能，独立出来做一台服务器，不会直接打到业务服务器上）

# 四，Nginx解决的问题

++++++++++++++++++++++++++++++++以下内容为快速整理+++++++++++++++++++++++++++

高并发、负载均衡、高可用、虚拟主机、伪静态、动静分离

### 高并发

### 负载均衡

负载均衡反向代理 ：

|  |
| --- |
| #返回的相应文件地址  location / {    #负载均衡反向代理  proxy\_pass http://myapp;    #返回根路径地址（相对路径:相对于/usr/local/nginx/）  root html;  #默认访问文件  index index.html index.htm;  }  upstream myapp {  server 192.168.1.171:8080 weight=1 max\_fails=2 fail\_timeout=30s;  server 192.168.1.172:8080 weight=1 max\_fails=2 fail\_timeout=30s;  } |

#### 负载均衡算法：

1. **轮询+weight**

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。某一台后端服务器宕机，Nginx 会自动将该服务器剔除出队列

|  |
| --- |
| 如下所示，10.0.0.88的访问比率要比10.0.0.77的访问比率高一倍。  upstream linuxidc {  server 10.0.0.77 weight=5;  server 10.0.0.88 weight=10;   } |

1. **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 favresin{  ip\_hash;  server 10.0.0.10:8080;  server 10.0.0.11:8080;   } |

1. **url\_hash**

按照访问的 URL 的 hash 结果分配请求，每个请求的 URL 会指向后端固定的某个服务器，可以在 Nginx 作为静态服务器的情况下提高缓存效率。

|  |
| --- |
| upstream resinserver{  server 10.0.0.10:7777;  server 10.0.0.11:8888;    hash $request\_uri;  hash\_method crc32;  } |

1. **least\_conn**

如果有些请求占用的时间很长，会导致其所在的后端负载较高。在这种场景下，把请求转发给连接数较少的后端，能够达到更好的负载均衡效果，这就是least\_conn算法。

least\_conn算法很简单，首选遍历后端集群，比较每个后端的conns/weight，选取该值最小的后端

1. **fair**

智能调整调度算法，动态的根据后端服务器的请求处理到响应的时间进行均衡分配。

响应时间短处理效率高的服务器分配到请求的概率高，响应时间长处理效率低的服务器分配到的请求少，它是结合了前两者的优点的一种调度算法。

但是需要注意的是 Nginx 默认不支持 fair 算法，如果要使用这种调度算法，请安装 upstream\_fair 模块。

### 高可用

### 虚拟主机

虚拟主机是一种特殊的软硬件技术，它可以将网络上的每一台计算机分成多个虚拟主机，每个虚拟主机可以独立对外提供www服务，这样就可以实现一台主机对外提供多个web服务，每个虚拟主机之间是独立的，互不影响的。

通过nginx可以实现虚拟主机的配置，nginx支持三种类型的虚拟主机配置：

1. 基于ip的虚拟主机， （一块主机绑定多个ip地址）
2. 基于域名的虚拟主机（servername）
3. 基于端口的虚拟主机（listen如果不写ip端口模式）

|  |
| --- |
| http{  server{  #表示一个虚拟主机  }  } |

**IP访问控制**

|  |
| --- |
| location {  deny IP /IP段  deny 192.168.1.109;  allow 192.168.1.0/24;192.168.0.0/16;192.0.0.0/8  } |

### 伪静态

### 动静分离

动静分离是将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问。

api接口服务化：动静分离之后，后端应用更为服务化，只需要通过提供api接口即可，可以为多个功能模块甚至是多个平台的功能使用，可以有效的节省后端人力，更便于功能维护。

前后端开发并行：前后端只需要关心接口协议即可，各自的开发相互不干扰，并行开发，并行自测，可以有效的提高开发时间，也可以有些的减少联调时间

减轻后端服务器压力，提高静态资源访问速度：后端不用再将模板渲染为html返回给用户端，且静态服务器可以采用更为专业的技术提高静态资源的访问速度。

|  |
| --- |
| location / {  proxy\_pass http://192.168.150.11:803;  }  location ~ .\*\.(gif|jpg|jpeg|png|bmp|swf|html|htm|css|js)$ {  root /var/data/www1/;  } |

### Session共享

#### Nginx + Memcached 可以实现session共享，但是实际不这么用

# 五，Https

大家都知道 HTTPS 比 HTTP 安全，也听说过与 HTTPS 协议相关的概念有 SSL 、非对称加密、 CA 证书等。

## 灵魂三拷问：

### 为什么用了 HTTPS 就是安全的?

### HTTPS 的底层原理如何实现?

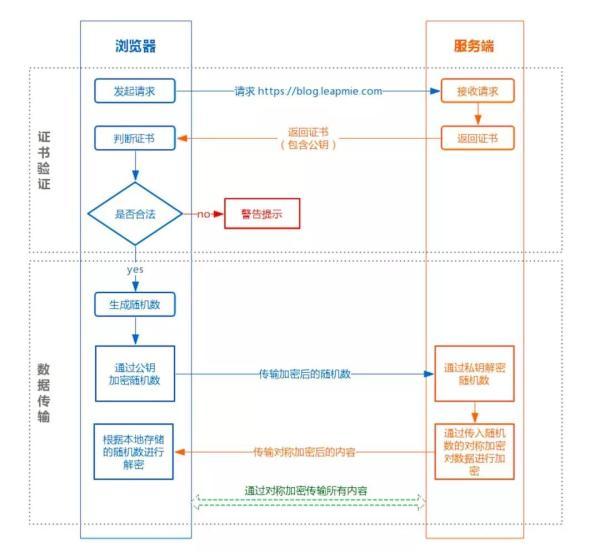
### 用了 HTTPS 就一定安全吗?

## HTTPS 的实现原理

大家可能都听说过 HTTPS 协议之所以是安全的是因为 HTTPS 协议会对传输的数据进行加密，而加密过程是使用了非对称加密实现。

但其实，HTTPS 在内容传输的加密上使用的是对称加密，非对称加密只作用在证书验证阶段。

HTTPS 的整体过程分为证书验证和数据传输阶段，具体的交互过程如下：



### 证书验证阶段：

1. 浏览器发起 HTTPS 请求。
2. 服务端返回 HTTPS 证书。
3. 客户端验证证书是否合法，如果不合法则提示告警。

### 数据传输阶段：

1. 当证书验证合法后，在本地生成随机数。
2. 通过公钥加密随机数，并把加密后的随机数传输到服务端。
3. 服务端通过私钥对随机数进行解密。
4. 服务端通过客户端传入的随机数构造对称加密算法，对返回结果内容进行加密后传输。

## ****为什么数据传输是用对称加密?****

首先，非对称加密的加解密效率是非常低的，而 HTTP 的应用场景中通常端与端之间存在大量的交互，非对称加密的效率是无法接受的。

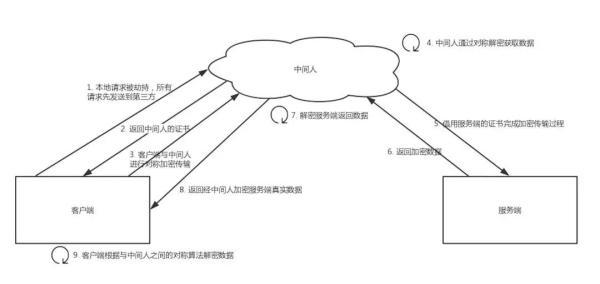
另外，在 HTTPS 的场景中只有服务端保存了私钥，一对公私钥只能实现单向的加解密，所以 HTTPS 中内容传输加密采取的是对称加密，而不是非对称加密。

## ****为什么需要 CA 认证机构颁发证书?****

HTTP 协议被认为不安全是因为传输过程容易被监听者勾线监听、伪造服务器，而 HTTPS 协议主要解决的便是网络传输的安全性问题。

首先我们假设不存在认证机构，任何人都可以制作证书，这带来的安全风险便是经典的“中间人攻击”问题。

“中间人攻击”的具体过程如下：



### 过程原理如下：

1. 本地请求被劫持(如 DNS 劫持等)，所有请求均发送到中间人的服务器。
2. 中间人服务器返回中间人自己的证书。
3. 客户端创建随机数，通过中间人证书的公钥对随机数加密后传送给中间人，然后凭随机数构造对称加密对传输内容进行加密传输。
4. 中间人因为拥有客户端的随机数，可以通过对称加密算法进行内容解密。
5. 中间人以客户端的请求内容再向正规网站发起请求。
6. 因为中间人与服务器的通信过程是合法的，正规网站通过建立的安全通道返回加密后的数据。
7. 中间人凭借与正规网站建立的对称加密算法对内容进行解密。
8. 中间人通过与客户端建立的对称加密算法对正规内容返回的数据进行加密传输。
9. 客户端通过与中间人建立的对称加密算法对返回结果数据进行解密。

由于缺少对证书的验证，所以客户端虽然发起的是 HTTPS 请求，但客户端完全不知道自己的网络已被拦截，传输内容被中间人全部窃取。

### ****浏览器是如何确保** CA 证书的合法性?**

证书包含什么信息?

1. 证书包含信息如下：
2. 颁发机构信息
3. 公钥
4. 公司信息
5. 域名
6. 有效期
7. 指纹
8. ......

### ****证书的合法性依据是什么?****

首先，权威机构是要有认证的，不是随便一个机构都有资格颁发证书，不然也不叫做权威机构。

另外，证书的可信性基于信任制，权威机构需要对其颁发的证书进行信用背书，只要是权威机构生成的证书，我们就认为是合法的。

所以权威机构会对申请者的信息进行审核，不同等级的权威机构对审核的要求也不一样，于是证书也分为免费的、便宜的和贵的。

### ****浏览器如何验证证书的合法性?****

浏览器发起 HTTPS 请求时，服务器会返回网站的 SSL 证书。

浏览器需要对证书做以下验证：

验证域名、有效期等信息是否正确。证书上都有包含这些信息，比较容易完成验证。

判断证书来源是否合法。每份签发证书都可以根据验证链查找到对应的根证书，操作系统、浏览器会在本地存储权威机构的根证书，利用本地根证书可以对对应机构签发证书完成来源验证。



1. 判断证书是否被篡改。需要与 CA 服务器进行校验。
2. 判断证书是否已吊销。通过 CRL(Certificate Revocation List 证书注销列表)和 OCSP(Online Certificate Status Protocol 在线证书状态协议)实现。

其中 OCSP 可用于第 3 步中以减少与 CA 服务器的交互，提高验证效率。

以上任意一步都满足的情况下浏览器才认为证书是合法的。

这里插一个我想了很久的但其实答案很简单的问题：既然证书是公开的，如果要发起中间人攻击，我在官网上下载一份证书作为我的服务器证书，那客户端肯定会认同这个证书是合法的，如何避免这种证书冒用的情况?

其实这就是非加密对称中公私钥的用处，虽然中间人可以得到证书，但私钥是无法获取的。

一份公钥是不可能推算出其对应的私钥，中间人即使拿到证书也无法伪装成合法服务端，因为无法对客户端传入的加密数据进行解密。