

# day08-rewrite-https-tomcat-cluster

---

- day08-rewrite-https-tomcat-cluster
  - 01.Nginx Rewrite重写
    - 1.Rewrite URL重写概述
    - 2.Rewrite URL重写模块
      - 2.1 if指令
      - 2.2 set指令
      - 2.3 return指令
      - 2.4 Rewrite重写语法
      - 2.5 break与last区别
      - 2.6 redirect与permanent区别
    - 3.Rewrite 生产案例实践
  - 02.Nginx HTTPS 实践
    - 1.HTTPS基本概述
    - 2.HTTPS实现原理
    - 3.HTTPS注意事项
    - 4.HTTPS单台配置实践
    - 5.HTTPS集群配置实践
    - 6.HTTPS优化配置实践
  - 03.Tomcat 快速入门
    - 1.Tomcat基本介绍
    - 2.Tomcat快速安装
    - 3.Tomcat配置文件
    - 4.Tomcat虚拟主机
    - 5.Tomcat管理页面
    - 6.Tomcat部署zrblog
  - 04.Tomcat 集群实战
    - 1.Tomcat+Nginx集群架构概述
    - 2.Tomcat+Nginx集群架构实战
    - 3.Tomcat+Nginx+Https实践
    - 4.Tomcat+Nginx+Redis实践

## 01.Nginx Rewrite重写

---

# 1.Rewrite URL重写概述

## 1.什么是rewrite

Rewrite主要实现url地址重写，以及url地址跳转。

就是将用户请求web服务器的URL地址重新修改为其他URL地址的过程。

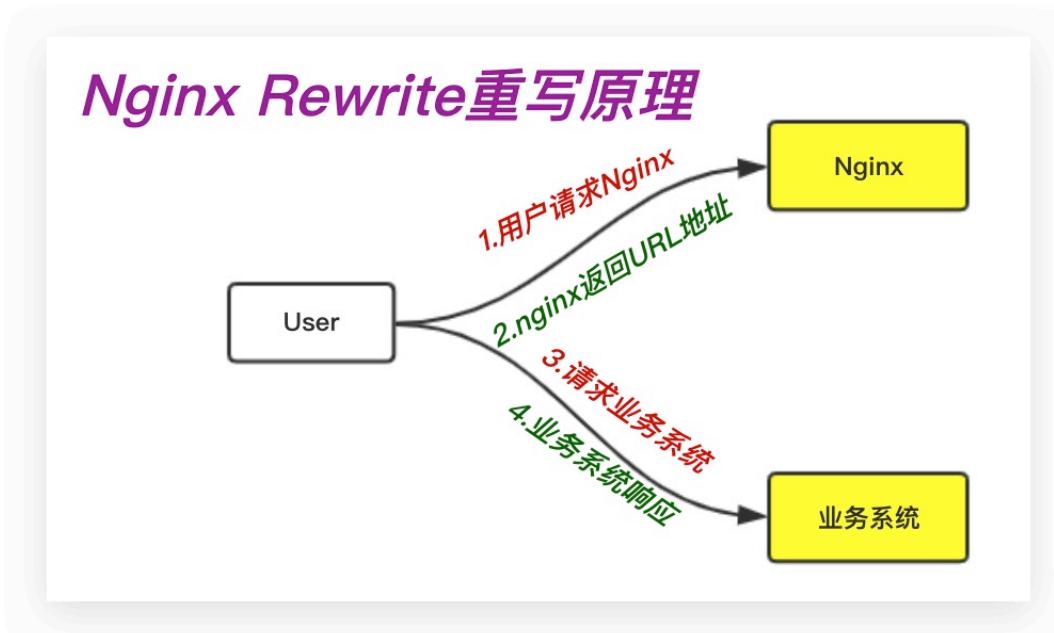
比如说京东,google、亚马逊都在使用:

域名	重写后域名
www.z.cn	www.amazon.cn
www.g.cn	www.google.cn
www.360buy.com	www.jd.com
58.com	bj.58.com

## 2.Rewrite使用场景

- 1.地址跳转，用户访问www.xlw.com/class这个URL时，将其定向至一个新的域名class.xlw.com
- 2.协议跳转，将用户通过http的请求协议重新跳转至https协议(实现https主要手段)。
- 3.URL静态化，将动态URL地址显示为静态URL的一种技术，能提高搜索引擎抓取，并且能减少动态URL对外暴露过多的参数。PS: Rewrite会轻微增加服务器负担。

## 3.Rewrite重写原理，如下图所示：



## 4.Rewrite重写相关模块。

set 设置变量

if 负责语句中的判断

return 返回返回值或URL

## 2.Rewrite URL重写模块

### 2.1 if指令

if指令语法介绍

```
Syntax: if (condition) { ... }
Default: -
Context: server, location
# ~ 模糊匹配
# ~* 不区分大小写的匹配
# !~ 不匹配
# = 精确匹配
```

需求1: 过滤Nginx 请求中包含 a1=3526 的http请求到 10.16.3.5 的 8080端口处理。

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    default_type application/json;
    root /code;

    location / {
        index index.html;
        if ( $request_uri ~* 'a1=\d{4}' ) {
            # proxy_pass http://10.16.3.5:8080;
        }
    }
}

#\d表示数字, {4,8}表示数字出现的次数是4到8次, 如gid=12345678就符合该条件。
if($request_uri ~* 'good=\d{4,8}/') {
    动作;
}

#测试
# curl -L test.oldxu.com/?id=123456
```

### 2.2 set指令

## 1.set指令语法

Syntax: `set $variable value;`  
Default: -  
Context: server, location, if

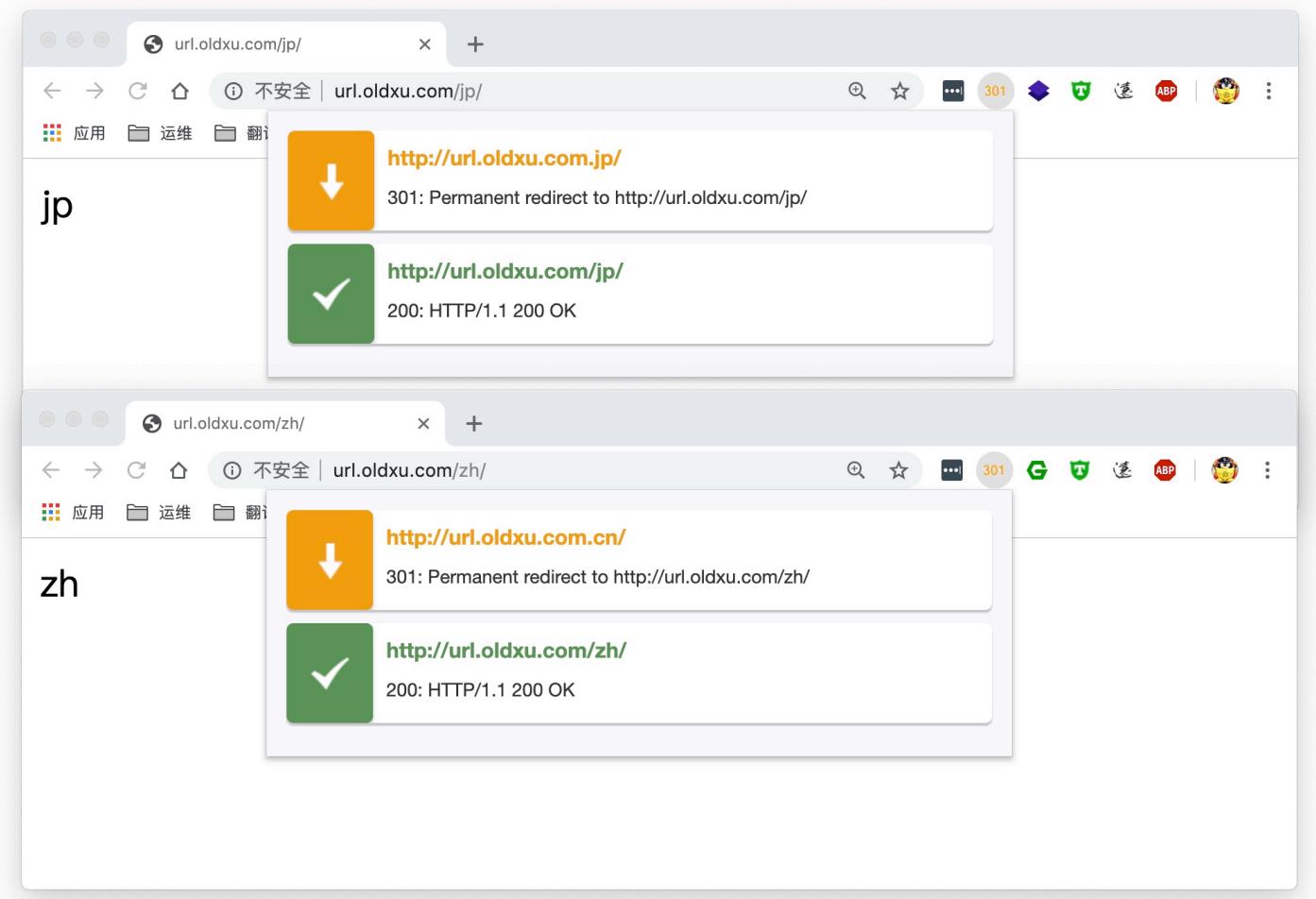
需求2: 将用户请求url.oldxu.com.zh跳转至url.oldxu.com/zh

需求3: 将用户请求url.oldxu.com.jp跳转至url.oldxu.com/jp

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com.zh url.oldxu.com.;
    location / {
        if ( $http_host ~* "zh" ) {
            set $language zh;
        }
        if ( $http_host ~* "jp" ) {
            set $language jp;
        }
        rewrite ^/$ http://url.oldxu.com/$language/ permanent;
    }
}

server {
    listen 80;
    server_name url.oldxu.com;
    location / {
        root /code;
        index index.html;
    }
}

#准备代码
[root@web01]# mkdir /code/jp /code/zh
[root@web01]# echo "zh" > /code/zh/index.html
[root@web01]# echo "jp" > /code/jp/index.html
[root@web01]# systemctl restart nginx
```



## 2.3 return指令

### 1. return数据返回语法

```
Syntax: return code;  
return code [text];  
return URL;  
Default: -  
Context: server, location, if
```

需求1: 如果用户使用IE浏览器访问url.oldxu.com，则返回段字符串。

```
[root@web01]# cat url.oldxu.com.conf  
server {  
    listen 80;  
    server_name url.oldxu.com;  
    root /code;  
    default_type text/html;  
    charset utf-8;  
  
    location / {
```

```
    index index.html;
    if ( $http_user_agent ~* "MSIE|firefox" ) {
        return 200 "请更换浏览器";
    }
}
}
```

需求2: 如果用户使用IE浏览器访问url.oldxu.com, 则返回500错误。

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;
    default_type text/html;
    charset utf-8;

    location / {
        index index.html;
        if ( $http_user_agent ~* "MSIE|firefox" ) {
            return 500;
        }
    }
}
```

需求3: 如果用户使用IE浏览器访问url.oldxu.com, 则直接跳转至Google浏览器下载站点。

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;
    default_type text/html;
    charset utf-8;

    location / {
        index index.html;
        if ( $http_user_agent ~* "MSIE|firefox" ) {
            return 302 https://www.xuliangwei.com;
        }
    }
}
```

## 2.4 Rewrite重写语法

`rewrite`主要是用来重写URL或者跳转URL的指令。

```
#rewrite表达式可以应用在server, location, if标签下
#      关键字    正则    替代内容    flag标记
Syntax: rewrite regex replacement [flag];
Default: --
Context: server, location, if

#flag
last          #本条规则匹配完成后，继续向下匹配新的Location URI规则
break         #本条规则匹配完成即终止，不再匹配后面的任何规则
redirect      #返回302临时重定向，地址栏会显示跳转后的地址
permanent    #返回301永久重定向，地址栏会显示跳转后的地址
```

## 2.5 break与last区别

### 1. 编写rewrite相关代码示例:

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    location / {
        rewrite /1.html /2.html;
        rewrite /2.html /3.html;
    }

    location /2.html {
        rewrite /2.html /a.html;
    }

    location /3.html {
        rewrite /3.html /b.html;
    }
}

#准备对应代码
[root@web01]# echo "1.html" >/code/1.html
[root@web01]# echo "2.html" >/code/2.html
[root@web01]# echo "3.html" >/code/3.html
[root@web01]# echo "a.html" >/code/a.html
[root@web01]# echo "b.html" >/code/b.html

#测试结果：当请求/1.html，最终将会访问/b.html
```

## 2. 为该代码增加break

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    location / {
        rewrite /1.html /2.html break;
        rewrite /2.html /3.html;
    }

    location /2.html {
        rewrite /2.html /a.html;
    }

    location /3.html {
        rewrite /3.html /b.html;
    }
}
```

#测试结果：当请求/1.html，最终会访问/2.html

#因为：在location{}内部，遇到break，本location{}内以及后面的所有location{}内的所有指令都不再执行。

## 3. 为该代码增加last

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    location / {
        rewrite /1.html /2.html last;
        rewrite /2.html /3.html;
    }

    location /2.html {
        rewrite /2.html /a.html;
    }

    location /3.html {
        rewrite /3.html /b.html;
    }
}
```

```
}
```

```
#测试结果：当请求/1.html，最终会访问/a.html  
#因为：在location{}内部，遇到last，本location{}内后续指令不再执行  
#而重写后的url会对所在的server{...}标签重新发起请求，从头到尾匹配一遍规则，那个匹配则执行哪个。
```

#### 4.break与last区别说明？

当rewrite规则遇到break后，本location{}与其他location{}的所有rewrite/return规则都不再执行。  
当rewrite规则遇到last后，本location{}里后续rewrite/return规则不执行，但重写后的url再次从头开始执行所有规则，哪个匹配执行哪个。

## 2.6 redirect与permanent区别

### 1.redirect与permanent区别对比示例

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    location / {
        rewrite /1.html /2.html redirect;
        rewrite /2.html /3.html;
    }
}
```

2.通过浏览器访问测试，会发现无论是permanent、还是redirect会进行跳转。

	<a href="http://url.oldxu.com/">http://url.oldxu.com/</a>	301: Permanent redirect to https://www.xuliangwei.com/
	<a href="https://www.xuliangwei.com/">https://www.xuliangwei.com/</a>	200: HTTP/1.1 200

	<a href="http://url.oldxu.com.cn/">http://url.oldxu.com.cn/</a>	302: Temporary redirect to https://www.xuliangwei.com/
	<a href="https://www.xuliangwei.com/">https://www.xuliangwei.com/</a>	200: HTTP/1.1 200

3. 那么**redirect**和**permanent**都是跳转，那**redirect**和**permanent**区别在哪呢？

Flag	跳转	状态码	排名情况
redirect	临时跳转	302	对旧网站无影响，新网站会有排名
permanent	永久跳转	301	新跳转网站有排名，旧网站排名会被清空

### 3. Rewrite 生产案例实践

需求1：根据用户浏览器请求头中携带的语言调度到不同的页面。

```
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    if ($http_accept_language ~* "zh-CN|zh") {
```

```
    set $language /zh;
}
if ($http_accept_language ~* "en") {
    set $language /en;
}
rewrite ^/$ /$language; #根据语言跳转对应的站点

location / {
    index index.html;
}
}
```

需求2: 用户通过手机设备访问 url.oldxu.com, 跳转至url.oldxu.com/m

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    if ($http_user_agent ~* "android|iphone|ipad") {
        rewrite ^/$ /m;
    }
}
```

需求3: 用户通过手机设备访问 url.oldxu.com 跳转至 m.oldxu.com

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    if ($http_user_agent ~* "android|iphone|ipad") {
        rewrite ^/$ http://m.oldxu.com;
    }
}

server {
    listen 80;
    server_name m.oldxu.com;
    root /data/m;

    location / {
        index index.html;
    }
}
```

```
}
```

需求4: 用户通过http协议请求, 能自动跳转至https协议。

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    rewrite ^(.*)$ https://$server_name$1 redirect;
    #return 302 https://$server_name$request_uri;
}
server {
    listen 443;
    server_name url.oldxu.com;
    ssl on;
}
```

需求5: 网站在维护过程中, 希望用户访问所有网站重定向至一个维护页面。

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    #配置示例
    rewrite ^(.*)$ /wh.html break;

    location / {
        index index.html;
    }
}
```

需求6: 当服务器遇到 403 403 502 等错误时, 自动转到临时维护的静态页。错误页面模板

```
[root@web01]# cat url.oldxu.com.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

    location / {
        index index.html;
```

```

}

#配置示例
error_page 404 403 502 = @tempdown;
location @tempdown {
    rewrite ^(.*)$ /wh.html break;
}
}

```

需求7: 公司网站在停机维护时, 指定的IP能够正常访问, 其他的IP跳转到维护页。

```

[root@web01 conf.d]# cat wh.conf
server {
    listen 80;
    server_name url.oldxu.com;
    root /code;

#1. 在server层下设定ip变量值为0
set $ip 0;
#2. 如果来源IP是10.0.0.101 102则设定变量为ip变量为1。
if ($remote_addr = "10.0.0.101|10.0.0.102") {
    set $ip 1;
}
#3. 如果来源IP不是10.0.0.101 102、则跳转至/code/wh.html这个页面, 否则不做任何处理
if ($ip = 0) {
    rewrite ^(.*)$ /wh.html break;
}
#-->如果想针对某个location进行操作, 则将如上配置写入location中即可

location / {
    index index.html;
}
}

```

需求8: 公司网站后台/admin, 只允许公司的出口公网IP可以访问, 其他的IP访问全部返回500, 或直接跳转至首页。

```

#限制访问来源IP
location /admin {
    set $ip 0;
    if ($remote_addr = "61.149.186.152|139.226.172.254" ) {
        set $ip 1;
    }
    if ($ip = 0){
        return 500;
    }
}

```

```
#rewrite /(.*)$ https://url.oldxu.com redirect;
}
}
```

## 02.Nginx HTTPS 实践

### 1. HTTPS基本概述

#### 1.为什么要使用HTTPS？

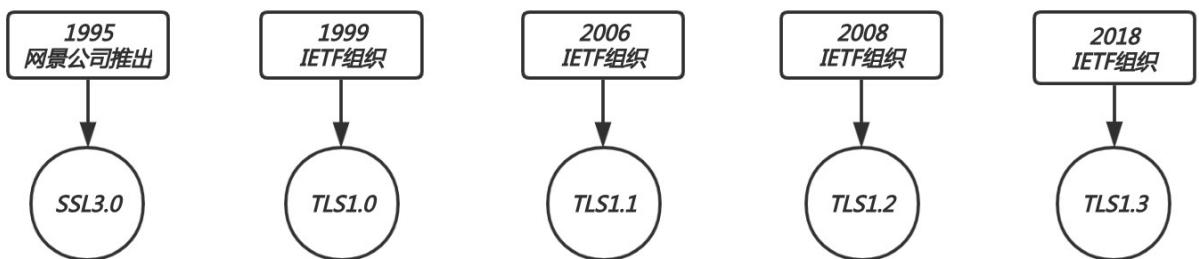
因为HTTP采用的是明文传输数据，那么在传输（账号密码、交易信息、等敏感数据）时不安全。容易遭到篡改，如果使用HTTPS协议，数据在传输过程中是加密的，能够有效避免网站传输时信息泄露。

#### 2.什么是HTTPS？

HTTPS安全的超文本传输协议，我们现在大部分站点都是通过HTTPS来实现站点数据安全。早期网景公司设计了SSL（Secure Socket Layer）安全套接层协议，主要用于对HTTP协议传输的数据进行加密。那如何将站点变成安全的HTTPS站点呢？我们需要了解SSL（Secure Socket Layer）协议。

而现在很多时候我们在使用TLS（Transport Layer Security）传输层安全协议。HTTP Over TLS

### SSL/TLS关系



#### 3.TLS/SSL是如何实现HTTP明文消息被加密的？

TLS/SSL协议工作在OSI七层模型中，应用层与传输层之间。

- 1.提供数据安全：保证数据不会被泄露。
- 2.提供数据的完整性：保证数据在传输过程中不会被篡改。
- 3.对应用层交给传输层的数据进行加密与解密。



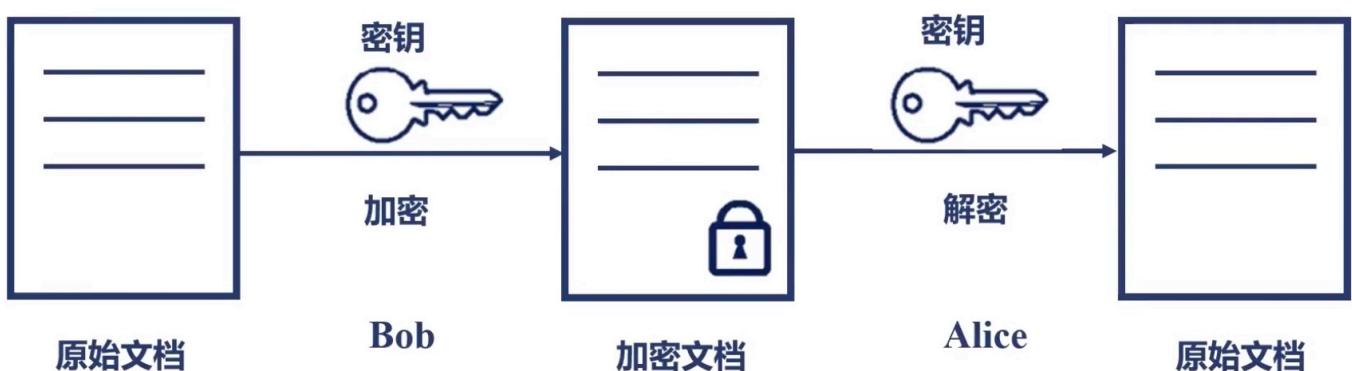
## 2. HTTPS实现原理

### 1. HTTPS加密模型，对称加密以及非对称加密

1) 对称加密，两个想通讯的人持有相同的秘钥，进行加密与解密。如下：

bob 将原始文档通过秘钥加密生成一个密文文档。

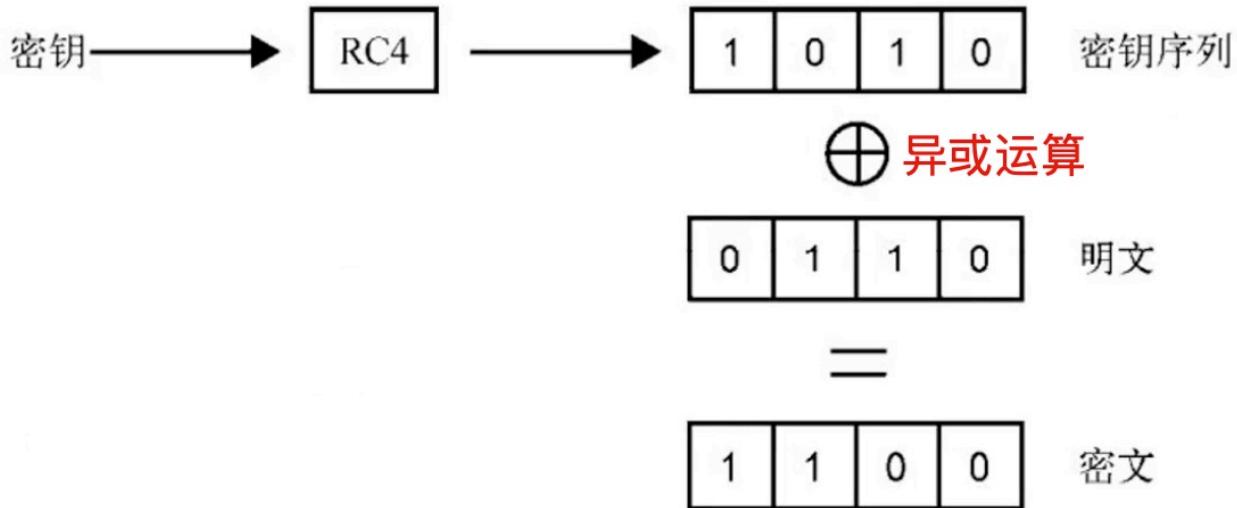
alice 拿到这个密文文档以后，它可以用这把秘钥还原为原始的明文文档。



对称加密究竟是如何实现的，我们可以以RC4这样一个对称加密序列算法来看一下。

加密：秘钥序列+明文=密文

解密：秘钥序列+密文=明文

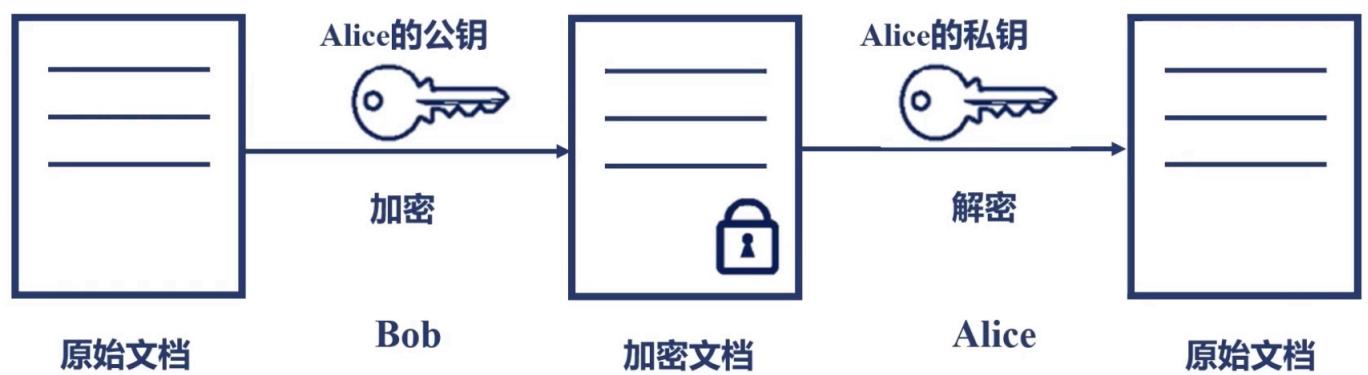


2) 非对称加密：它根据一个数学原理，它会生成一对秘钥（公钥和私钥）公钥加密：私钥解密，只能实现单方向。

私钥：私钥自己使用，不对外开放。

公钥：公钥给大家使用，对外开放。

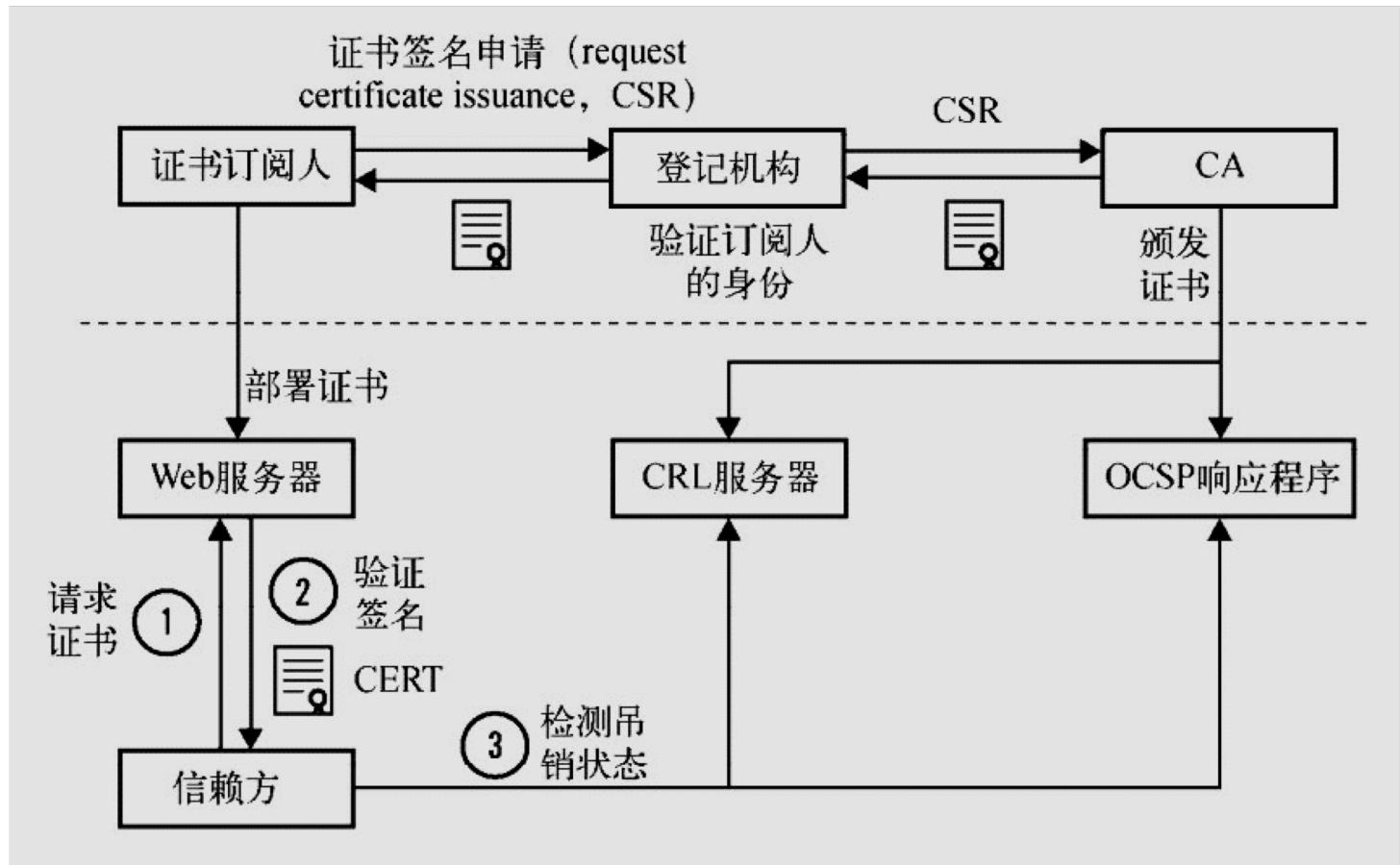
比如：alice有一对公钥和私钥，他可以将公钥发布给任何人。假设Bob是其中一个，当Bob要传递一份加密文档给Alice，那么Bob就可以用Alice的公钥进行加密，Alice收到密文文档后通过自己的私钥进行解密，获取原始文档。



注意：Alice必须知道Bob就是Bob，也就是它收到的信息必须是Bob发来的，那么这个信任问题，在多方通讯的过程中，必须有一个公信机构来验证双方的身份，那么这个机构就是我们的CA机构。

## 2. Https 通讯过程是如何验证双方的身份？

CA架构是可信任组织架构，主要用来颁发证书。那CA机构又是如何申请和颁发证书的呢？

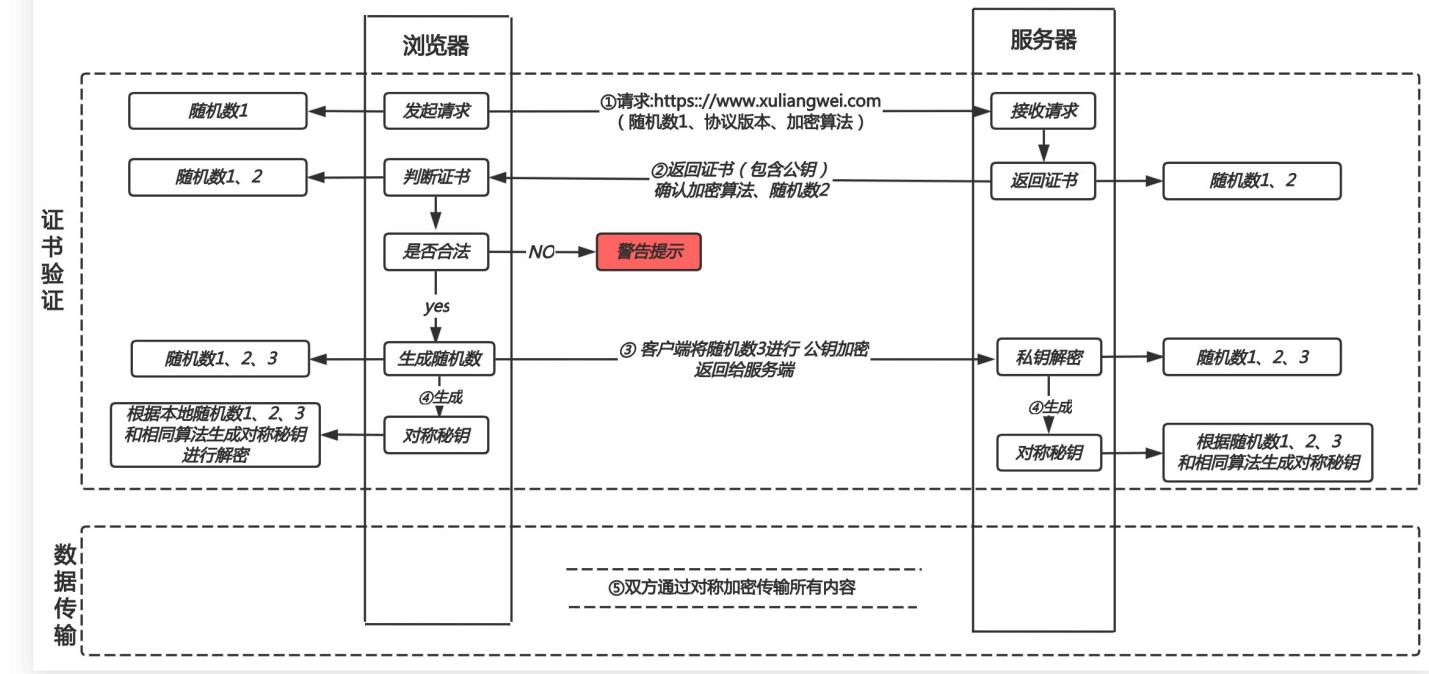


我们首先需要申请证书，需要进行登记，登记我是谁，我是什么组织，我想做什么，到了登记机构在通过CSR发给CA，CA中心通过后，CA中心会生成一对公钥和私钥，那么公钥会在CA证书链中保存，公钥和私钥证书订阅人拿到后，会将其部署在WEB服务器上

1. 当浏览器访问我们的https站点时，它会去请求我们的证书
2. Nginx会将我们的公钥证书回传给浏览器。
3. 浏览器会去验证我们的证书是否是合法的、是否是有效的。
4. CA机构会将过期的证书放置在CRL服务器，那么CRL服务的验证效率是非常差的，所以CA又推出了OCSP响应程序，OCSP响应程序可以查询指定的一个证书是否过期，所以浏览器可以直接查询OCSP响应程序，但OCSP响应程序性能还不是很高。
5. Nginx会有一个OCSP的开关，当我们开启后，Nginx会主动上OCSP上查询，这样大量的客户端直接从Nginx获取，证书是否有效。

### 3. HTTPS如何实现加密与解密

## Https加密解密原理



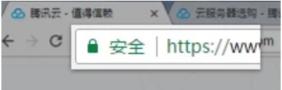
HTTPS 加密过程，HTTPS 采用混合加密算法，即对称加密、和非对称加密

通信前准备工作：申请域名对应的证书，并将其部署在Nginx服务器中。

- 1) 第一步客户端向服务端发送 *Client Hello* 消息，这个消息里包含了一个客户端生成的随机数 *Random1*、客户端支持的加密套件 和 客户端支持 *TLS* 协议版本等信息。
- 2) 服务端会向客户端发送 *Server Hello* 消息。返回自己的公钥证书、挑选一个合适的加密套件、另外还会生成一份随机数 *Random2* 推送给客户端。至此客户端和服务端都拥有了两个随机数 (*Random1 + Random2*)
- 3) 客户端收到服务端传来的公钥证书后，先从 CA 验证该证书的合法性 (CA 公钥去解密公钥证书)，验证通过后取出证书中的服务端公钥，再生成一个随机数 *Random3*，再用服务端公钥非对称加密 *Random3*。
- 4) 服务端用自己的私钥解出客户端生成的 *Random3*。至此，客户端和服务端都拥有 *Random1 + Random2 + Random3*，两边根据同样的算法生成一份秘钥，握手结束后的应用层数据都是使用这个秘钥进行对称加密。

## 3. HTTPS注意事项

### 1. https证书都有哪些类型？

对比	域名型 DV	企业型 OV	增强型 EV
绿色地址栏 (以chrome为例)	 小锁标记 + https	 小锁标记 + https	 小锁标记 + https + 公司名称
一般用途	个人站点和应用；简单的Https加密需求	电子商务站点和应用；中小型企业站点	大型金融平台；大型企业和政府机构站点
审核内容	域名所有权验证	全面的企业身份验证；域名所有权验证	最高等级的企业身份验证；域名所有权验证
颁发时长	10分钟-24小时	3-5个工作日	5-7个工作日
单次申请年限	1年	1-2年	1-2年
赔付保障金	—	125-175万美金	150-175万美金

## 2. HTTPS证书购买指南

保护1个域名 www

保护5个域名 www images cdn test m

通配符域名 \*.oldxu.com

## 3. HTTPS颜色提示

Https不支持续费，证书到期需重新申请并进行替换。

Https不支持三级域名解析，如 test.m.xu.com。

Https显示绿色，说明整个网站的url都是https的，并且都是安全的。

Https显示黄色，说明网站代码中有部分URL地址是http不安全协议的。

Https显示红色，要么证书是假的，要么证书已经过期。

## 4. HTTPS单台配置实践

### 1. 环境准备

```
[root@web01 ~]# mkdir -p /etc/nginx/ssl_key
[root@web01 ~]# cd /etc/nginx/ssl_key
```

### 2. 使用openssl命令充当CA权威机构创建证书(生产不使用此方式生成证书，不被互联网认可的黑户证书)

```
[root@web01 ssh_key]# openssl genrsa -idea -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
```

```
#记住配置密码，我这里是1234
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

### 3.生成自签证书，同时去掉私钥的密码

```
[root@web01 ssl_key]# openssl req -days 36500 -x509 \
-sha256 -nodes -newkey rsa:2048 -keyout server.key -out server.crt

Country Name (2 letter code) [XX]:CN          #国家
State or Province Name (full name) []:WH      #省
Locality Name (eg, city) [Default City]:WH    #城市
Organization Name (eg, company) [Default Company Ltd]:edu  #公司
Organizational Unit Name (eg, section) []: oldxu   #单位
Common Name (eg, your name or your servers hostname) []: s.oldxu.com #服务器主机名称
Email Address []:oldxu@qq.com

# req -->用于创建新的证书
# new -->表示创建的是新证书
# x509 -->表示定义证书的格式为标准格式
# key -->表示调用的私钥文件信息
# out -->表示输出证书文件信息
# days -->表示证书的有效期
```

### 4.证书申请完成后需要了解Nginx如何配置Https

```
#官方示例

worker_processes auto;

http {
    ...
    server {
        listen          443 ssl;
        keepalive_timeout 70;

        ssl_protocols    TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers      AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
        ssl_certificate   /usr/local/nginx/conf/cert.pem;
        ssl_certificate_key /usr/local/nginx/conf/cert.key;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 10m;
        ...
    }
}
```

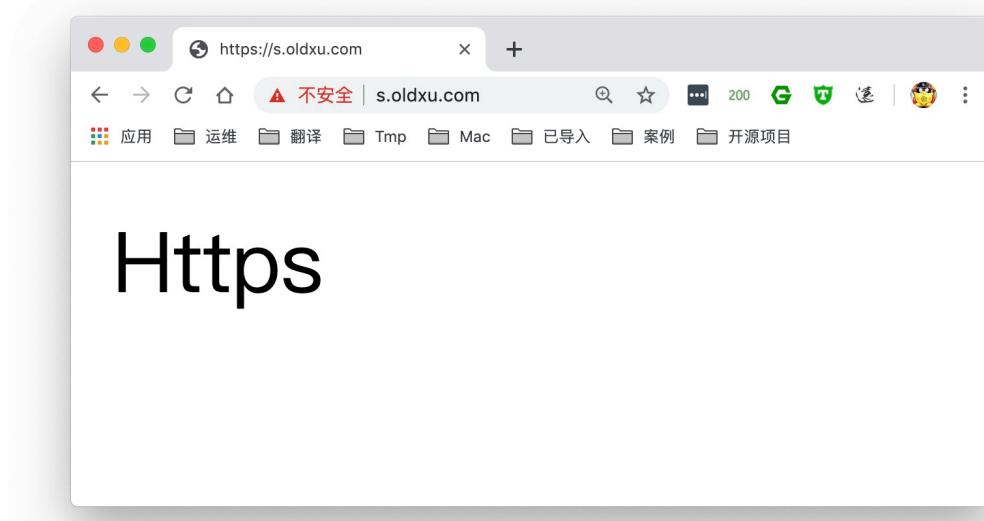
## 5. 配置Nginx配置Https实例

```
[root@web01 ~]# cat s.oldxu.com.conf
server {
    listen 443 ssl;
    server_name s.oldxu.com;
    ssl on;
    ssl_certificate    ssl_key/server.crt;
    ssl_certificate_key    ssl_key/server.key;
    location / {
        root /code;
        index index.html;
    }
}
```

#准备对应的站点目录，并重启Nginx服务

```
[root@web01]# mkdir -p /code
[root@web01]# echo "Https" > /code/index.html
[root@web01]# systemctl restart nginx
```

## 6. 浏览器输入 https://s.oldxu.com 访问，由于该证书非第三方权威机构颁发，而是我们自己签发的，所以浏览器会警告



## 7. 以上配置如果用户忘记在浏览器地址栏输入 https:// 那么将不会跳转至https，建议配置将用户访问 http 请求强制跳转https

```
[root@Nginx ~]# cat /etc/nginx/conf.d/ssl.conf
server {
    listen 443;
    server_name s.oldxu.com;
    ssl on;
```

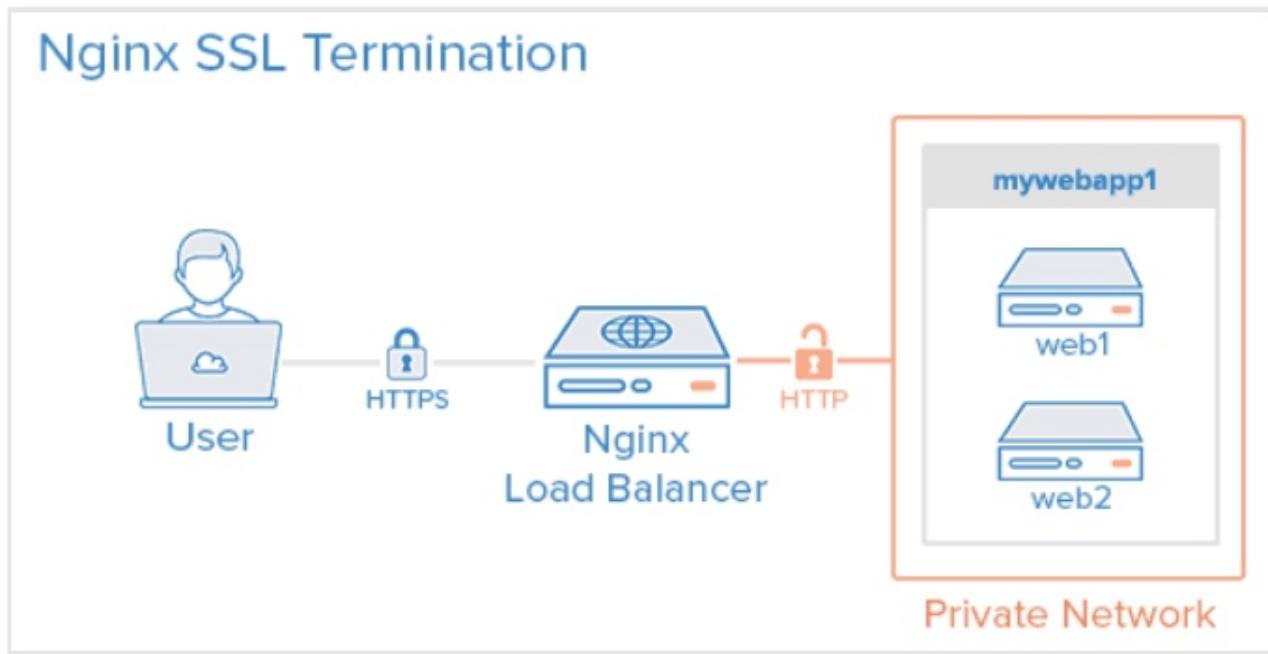
```

ssl_certificate    ssl_key/server.crt;
ssl_certificate_key  ssl_key/server.key;
location / {
    root /code;
    index index.html;
}
server {
    listen 80;
    server_name s.oldxu.com;
    return 302 https://$server_name$request_uri;
}

```

## 5. HTTPS集群配置实践

实战Nginx负载均衡+Nginx WEB配置HTTPS安全



### 1) 环境准备

主机名	外网IP(NAT)	内网IP(LAN)	角色
lb01	eth0:10.0.0.5	eth1:172.16.1.5	nginx-proxy
web01	eth0:10.0.0.7	eth1:172.16.1.7	nginx-web01
web02	eth0:10.0.0.8	eth1:172.16.1.8	nginx-web02

2) 配置后端两台web 节点监听80端口, 如已配置则无需修改

```
[root@web01 conf.d]# cat s.oldxu.com.conf
server {
    listen 80;
    server_name s.oldxu.com;
    root /code/wordpress;

    location / {
        index index.html;
    }
}
```

### 3) 配置第二台web节点

```
[root@web02 conf.d]# cat s.oldxu.com.conf
server {
    listen 80;
    server_name s.oldxu.com;
    root /code/wordpress;

    location / {
        index index.html;
    }
}
```

### 4) 重启两台后端web节点Nginx

```
[root@web01 ~]# systemctl restart nginx
[root@web02 ~]# systemctl restart nginx
```

### 5) Nginx负载均衡先生成证书

```
[root@lb01 ~]# mkdir /etc/nginx/ssl_key -p
[root@lb01 ~]# cd /etc/nginx/ssl_key
[root@lb01 ~]# openssl genrsa -idea -out server.key 2048
[root@lb01 ~]# openssl req -days 36500 -x509 -sha256 \
-nodes -newkey rsa:2048 -keyout server.key -out server.crt
```

### 6) Nginx负载均衡配置文件如下

```
[root@lb01 ~]# cat /etc/nginx/conf.d/proxy.conf
# 定义后端资源池
upstream site {
```

```

server 172.16.1.7:80 max_fails=2 fail_timeout=10s;
server 172.16.1.8:80 max_fails=2 fail_timeout=10s;
}

# https配置
server {
    listen 443;
    server_name s.oldxu.com;
    ssl on;
    ssl_certificate ssl_key/server.crt;
    ssl_certificate_key ssl_key/server.key;
    location / {
        proxy_pass http://site;
        include proxy_params;
    }
}
# 用户http请求跳转至https
server {
    listen 80;
    server_name s.oldxu.com;
    return 302 https://$server_name$request_uri;
}

```

## 7) 重启Nginx 负载均衡

```

[root@lb01 ~]# nginx -t
[root@lb01 ~]# systemctl restart nginx

```

PS: 如果要将wordpress的http方式改造为https方式，可能会造成页面加载不成功或者无法登陆。

```

#web 节点增加此参数
location ~ \.php$ {
    ...
    fastcgi_param HTTPS on;
    ...
}

```

## 6. HTTPS优化配置实践

SSL的运行计算需要消耗额外的CPU资源，SSL通讯过程中『握手』阶段的运算最占用CPU资源，有如下几个方面可以进行调整与优化。

1. 设置worker进程数设置为等于CPU处理器的核心数。 `worker_processes auto;`

2. 启用 `keepalive` 长连接，一个连接发送更多个请求。
3. 启用 `shared` 会话缓存，所有 `worker` 工作进程之间共享的缓存，避免进行多次 SSL 『握手』。
4. 禁用 `builtin` 内置于 OpenSSL 中的缓存，仅能供一个 `worker` 工作进程使用。[使用 `shared` 缓存即禁止 `builtin`]

```
worker_processes auto;

http {

    ...

    server {
        listen 443 ssl;
        server_name www.example.com;
        ssl_certificate      www.example.com.crt;
        ssl_certificate_key  www.example.com.key;
        ssl_protocols         TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;           #Nginx决定使用哪些协议与浏览器进行通讯

        keepalive_timeout 70;                  #设置长连接
        #建立握手后如果连接断开，在session_timeout时间内再次连接，无需再次建立握手，可直接复用之间
        #缓存的连接。
        ssl_session_cache shared:SSL:10m;     #1M缓存空间能存储 4000 个会话数量
        ssl_session_timeout 1440m;            #配置会话超时时间（默认5分钟）
    }
}
```

## 03.Tomcat 快速入门

### 1.Tomcat基本介绍

#### 1.JVM基本介绍

JAVA业务都是运行在java虚拟机上的，java虚拟机简称为JVM（java Virtual Machine）  
所谓虚拟机是指：通过软件模拟具有完整硬件系统的功能、运行在一个完全隔离环境中的计算机系统。

#### 2.为什么JAVA需要JVM虚拟机？

像早期的C语言不支持跨平台，因为C语言要想在windows、linux、Mac上运行，需要进行分别编译。那么在linux上有非常多的优秀软件，如果需要在Windows上使用则需要重新进行编译。移植性很差。

而JAVA则不同，JAVA是可以跨平台、只需要将源代码进行一次编译，能够在多处运行。

## 那 JAVA 是怎么做到?

它只需要在Windows、Linux系统上运行一个jvm，这样我们能将java 编译好的war包在Windows 和 Linux平台运行起来，无需我们重复编译。而JVM是由jre 提供。(JAVA运行环境java runtime Environment )

## 3.JAVA环境JRE和JDK那么区别在哪?

jre是java的运行环境，会包含jvm

jdk是java的开发环境，会包含java的运行环境jre。

如果说单纯的运行java代码，只需要jre足够，但如果需要提供开发环境以及运行环境则需要jdk。

## 4.什么是Tomcat?

Tomcat 和此前学习过的Nginx类似，也是一个Web服务器软件。

只不过 Tomcat 是基于 JAVA 开发的 WEB 服务，主要解析Java代码。

## 2.Tomcat与Nginx有什么区别?

Nginx仅支持静态资源解析，而Tomcat支持解析java开发的web应用，还支持解析静态资源(效率不高)。

Nginx适合做前端负载均衡，而Tomcat适合做后端应用服务处理。

通常情况下，企业会使用 Nginx+tomcat 结合，由Nginx处理静态资源，Tomcat处理动态资源。

# 2.Tomcat快速安装

## 1.yum方式安装

```
[root@es-node1 ~]# yum install java -y
[root@es-node1 ~]# mkdir /soft && cd /soft
[root@es-node1 soft]# wget http://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-9/v9.0.26/bin/apache-tomcat-9.0.26.tar.gz
[root@es-node1 soft]# tar xf apache-tomcat-9.0.26.tar.gz
[root@es-node1 soft]# ln -s /soft/apache-tomcat-9.0.26 /soft/tomcat
```

## 2.二进制方式安装

```
[root@es-node1 ~]# mkdir /soft/
[root@es-node1 ~]# tar xf jdk-8u60-linux-x64.tar.gz -C /app/
[root@es-node1 ~]# ln -s /soft/jdk1.8.0_60 /soft/jdk
[root@es-node1 ~]# cat >> /etc/profile << EOF
export JAVA_HOME=/soft/jdk
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib:$JAVA_HOME/lib/tools.jar
```

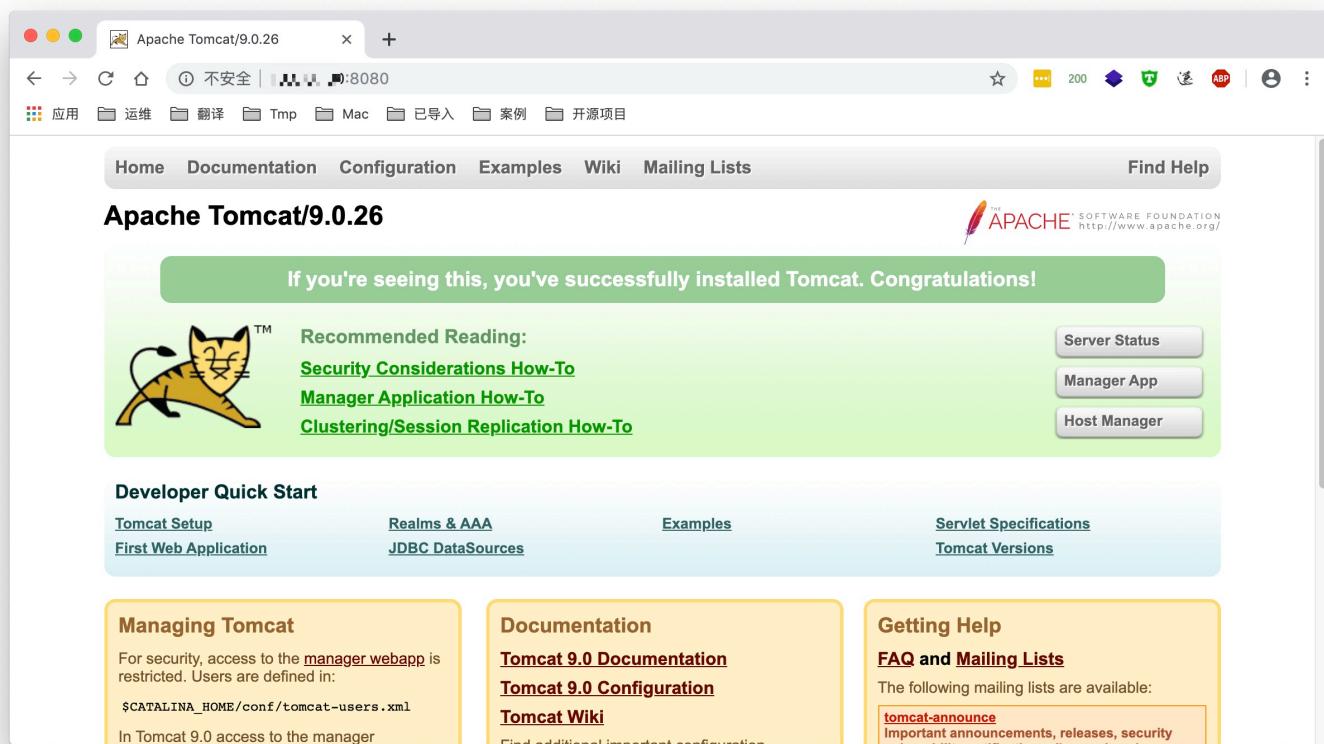
EOF

```
[root@es-node1 ~]# tar xf apache-tomcat-9.0.26.tar.gz -C /soft
[root@es-node1 ~]# /soft/apache-tomcat-9.0.26/bin/startup.sh
```

### 3. 启动 Tomcat 服务

```
[root@es-node1 soft]# /soft/tomcat/bin/startup.sh
[root@es-node1 soft]# netstat -lntp | grep java
tcp6      0      0 127.0.0.1:8005          ::::*                  LISTEN      370
4/java
tcp6      0      0 :::8009              ::::*                  LISTEN      370
4/java
tcp6      0      0 :::8080              ::::*                  LISTEN      370
4/java
```

### 4. 通过浏览器访问 <http://IP:8080> 测试访问



### 5. 为 tomcat 建立 systemd 启动单元，控制 Tomcat 启动和停止。

```
[root@es-node1 ~]# vim /usr/lib/systemd/system/tomcat.service
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target remote-fs.target nss-lookup.target

[Service]
```

```
Type=forking
Environment=CATALINA_HOME=/soft/tomcat
Environment=CATALINA_BASE=/soft/tomcat
ExecStart=/soft/tomcat/bin/startup.sh
ExecStop=/soft/tomcat/bin/shutdown.sh
```

```
[Install]
WantedBy=multi-user.target
```

```
#加载启动项
[root@es-node1 ~]# systemctl daemon-reload
# 加入开机启动项
[root@es-node1 ~]# systemctl enable tomcat
[root@es-node1 ~]# systemctl start tomcat
[root@es-node1 ~]# systemctl stop tomcat
```

## 3.Tomcat配置文件

### 1.Tomcat目录结构介绍

```
[root@es-node1 soft]# ll /soft/tomcat/
总用量 124
drwxr-x--- 2 root root 4096 9月 26 20:11 bin      #主要包含启动和关闭tomcat脚本，以及依赖的jar文件
drwx----- 3 root root 254 9月 26 20:12 conf     #tomcat配置文件目录
drwxr-x--- 2 root root 4096 9月 26 20:11 lib      #tomcat运行需要加载的jar包
drwxr-x--- 2 root root 197 9月 26 20:12 logs    #tomcat在运行过程中产生的日志文件
drwxr-x--- 2 root root 30  9月 26 20:11 temp    #tomcat存放临时文件
drwxr-x--- 7 root root 81  9月 16 23:52 webapps #tomcat默认站点目录
drwxr-x--- 3 root root 22  9月 26 20:12 work    #tomcat运行时产生的缓存文件
```

### 2.Tomcat主配置文件->介绍

```
[root@es-node1 conf]# cat server.xml
<?xml version="1.0" encoding="UTF-8"?>

<!-- server表示一个tomcat实例，通过8005远程调用可以关闭tomcat-->
<Server port="8005" shutdown="SHUTDOWN">

<!--监听器-->
<Listener className="org.apache.catalina.startup.VersionLoggerListener" />
<Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on"
/>
<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"
```

```
/>
<Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener"
/>

<!--全局资源-->
<GlobalNamingResources>
    <!--base认证模块->所有站点都能使用->conf/tomcat-users.xml文件定义-->
    <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>

<!--http连接器->处理请求与响应-->
<Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="8443" />

    <!--ajp连接器->apache代理tomcat使用mod_jk需要该协议 -->
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <!--engine引擎-->
    <Engine name="Catalina" defaultHost="localhost">

        <!--多个tomcat配置相同cluster 进行session共享的一种解决方案-->
        <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>

        <!--调用认证配置，引入UserDatabase资源定义的用户名及密码进行base认证-->
        <Realm className="org.apache.catalina.realm.LockOutRealm">
            <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase"/>
        </Realm>

        <!--默认网站站点-->
        <Host name="localhost" appBase="webapps"
            unpackWARs="true" autoDeploy="true">
            <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                prefix="localhost_access_log" suffix=".txt"
                pattern="%h %l %u %t \"%r\" %s %b" />
        </Host>
    </Engine>
</Service>
</Server>
```

### 3.Tomcat主配置文件>图解

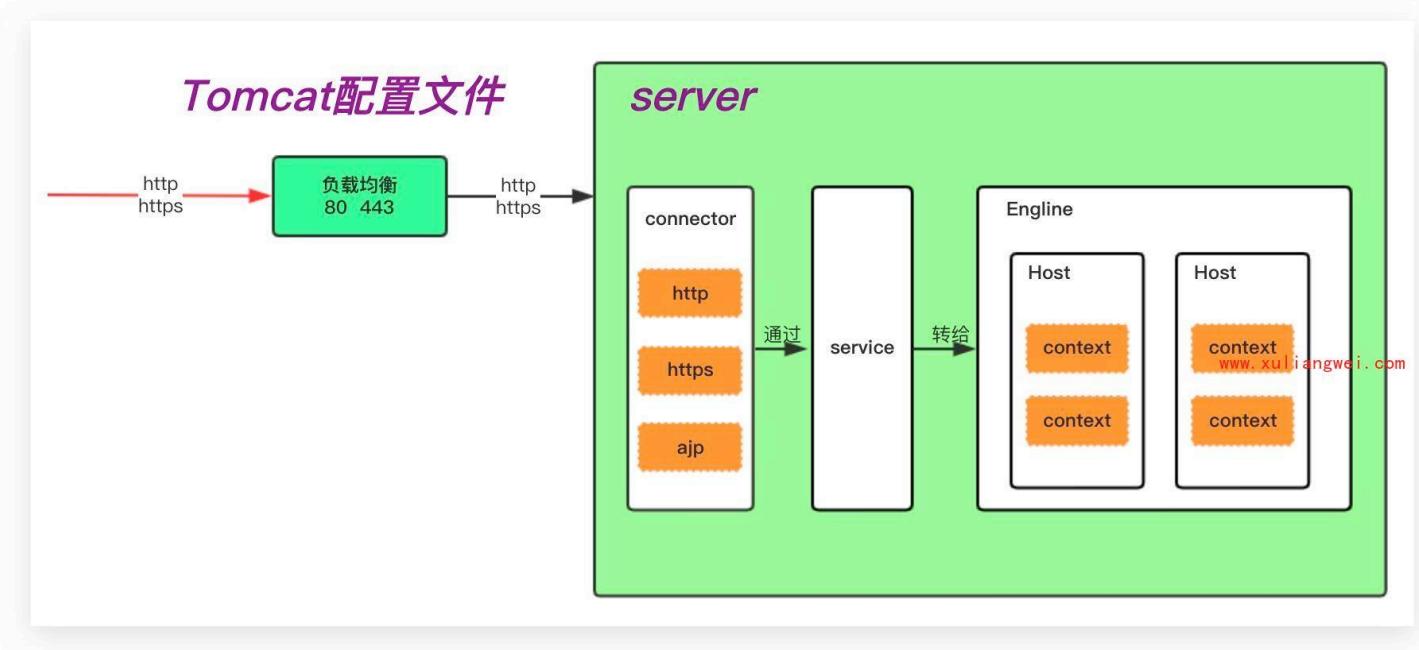
一个server表示一个tomcat实例（当然也可以有多个实例）

一个server中包含多个Connector连接器，Connector的主要功能是接受、响应用户请求。

service的作用是：将connector关联至engine(catalina引擎)

一个host就是一个站点，类似于nginx的多站点

context类似于nginx中location的概念



### Tomcat的HTTP请求过程

用户发出一个请求，如 `http://tomcat.oldxu.com:8080/index.jsp`

Connector发现是http/1.1协议，而且还是8080端口，于是就把请求接收后交给符合条件的Engine

Engine通过请求中的主机名tomcat.oldxu.com查找满足条件的虚拟主机(Host)

找到后就去此虚拟主机指定的appBase（代码存放的目录）最后将解析产生的结果返回给用户。

## tomcat目录结构与配置文件

## 4.Tomcat虚拟主机

虚拟主机定义：在一台web服务器软件上运行多个网站

```
[root@es-node1 ~]# vim /soft/tomcat/conf/server.xml
<Host name="tomcat1.oldxu.com" appBase="/code1"
      unpackWARs="true" autoDeploy="true">
  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="log"
        prefix="tomcat1_access_log" suffix=".txt"
        pattern="%h %l %u %t &"%r&" %s %b" />
```

```

</Host>

<Host name="tomcat2.oldxu.com" appBase="/code2"
      unpackWARs="true" autoDeploy="true">

    <Context docBase="/code/zh" path="/zh" reloadable="true"/>
    <Valve className="org.apache.catalina.valves.AccessLogValve" directory="log
s"
          prefix="zrlog_access_log" suffix=".txt"
          pattern="%h %l %u "%r"; %s %b" />
</Host>

#context的使用方式(如果设定了context, 确没有创建目录则会造成无法启动Tomcat)
<Context docBase="/code/zh" path="/zh" reloadable="true"/>
#context: 可以理解是nginx的location
#path: 虚拟目录
#docBase: 网页实际存放位置的根目录, 映射为path虚拟目录
#reloadable="true": 修改了jsp文件后不需要重启就可以实现显示的同步
#简单理解: 访问http://tomcat.oldxu.com/zh --映射-->/code/zh

```

## 5.Tomcat管理页面

当我们点击Tomcat默认站点Server Status、或者manager APP时，会出现403错误，提示需要配置Basic认证才能访问管理页面，或状态页面。

### 1. 配置conf/tomcat-users.xml

```

<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="tomcat" password="123456" roles="manager-gui,admin-gui"/>

```

2. 由于Tomcat仅允许本地127.0.0.1进行basic认证。如果需要其他网段通过basic认证，还需要配置允许访问规则。找到 webapps/项目/META-INF/context.xml

```
[root@es-node1 tomcat]# vim 项目目录下/META-INF/context.xml
allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
#修改为
allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|10\.0\.0\.\d+" />

#Tomcat默认管理页面
# vim /soft/tomcat/webapps/host-manager/META-INF/context.xml
# vim /soft/tomcat/webapps/manager/META-INF/context.xml
```

## 6.Tomcat部署zrblog

1.先为tomcat新增一个Host虚拟主机

```
[root@es-node1 ~]# vim /soft/tomcat/conf/server.xml
<Host name="zrlog.oldxu.com" appBase="/code"
      unpackWARs="true" autoDeploy="true">
  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
        prefix="zrlog_access_log" suffix=".txt"
        pattern="%h %l %u %t "%r" %s %b" />
</Host>
```

2.上传zrlog，并重启tomcat

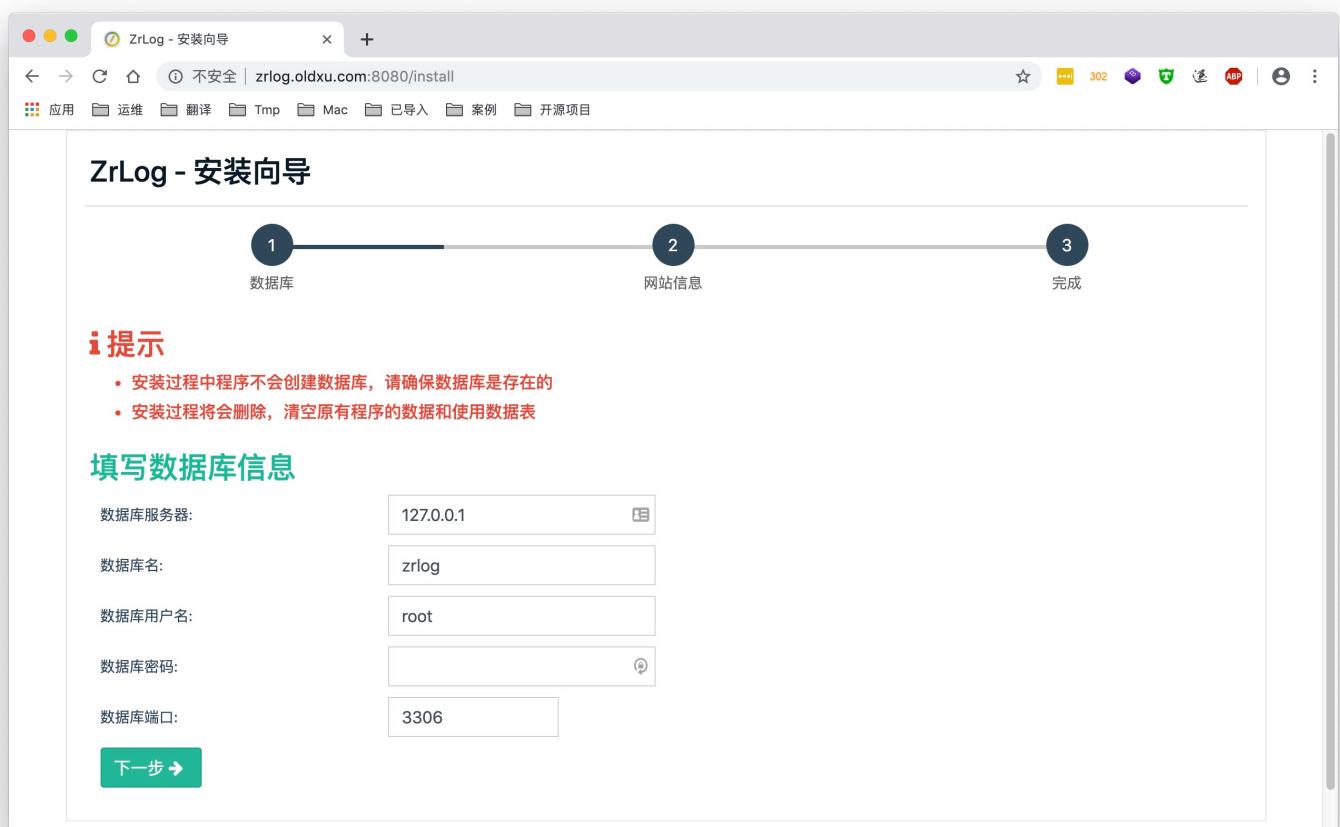
```
[root@es-node1 code]# /soft/tomcat/bin/shutdown.sh && /soft/tomcat/bin/startup.sh
```

3.由于zrlog需要数据库支持，所以需要安装一个mariadb，当然也可以直接使用远程数据库

```
[root@es-node1 code]# yum install mariadb mariadb-server -y

#使用远程数据库需要配置授权
mysql> create database zrlog charset utf8;
mysql> grant all privileges on *.* to tomcat@'%' identified by 'oldxu.com';
```

4.通过浏览器访问，需要带上8080端口



## 5. 安装后的效果



## 6. 部署多节点Tomcat-->mysql

#1. 安装jdk

```
[root@web02 ~]# yum install java -y
```

#2. 安装tomcat 部署代码 (scp)

```
#在web01上操作  
[root@web01 ~]# scp -rp /soft root@172.16.1.8:/  
[root@web01 ~]# scp -rp /code/zrLog root@172.16.1.8:/code/
```

```
#在web02上操作  
[root@web02 soft]# rm -rf /soft/tomcat/  
[root@web02 soft]# ln -s /soft/apache-tomcat-9.0.26 /soft/tomcat  
[root@web02 soft]# /soft/tomcat/bin/startup.sh
```

#3. 修改域名解析, 测试访问

## 7. 部署多节点Tomcat-->NFS

### #1. 安装NFS

```
[root@nfs ~]# groupadd -g 666 www  
[root@nfs ~]# useradd -u666 -g666 www  
[root@nfs ~]# yum install nfs-utils -y  
[root@nfs ~]# cat /etc/exports  
/data/zrlog 172.16.1.0/24(rw,sync,all_squash,anonuid=666,anongid=666)  
[root@nfs ~]# mkdir /data/zrLog  
[root@nfs ~]# systemctl restart nfs
```

### #2. 找到图片资源 推送图片资源至NFS

```
[root@web01 ~]# scp -rp /code/zrLog/ROOT/attached/* root@172.16.1.31:/data/zrLog/  
[root@nfs ~]# chown -R www.www /data/zrLog/ #重新授权
```

### #3. 多web节点挂载

```
# mount -t nfs 172.16.1.31:/data/zrLog/ /code/zrLog/ROOT/attached/
```

# 04.Tomcat 集群实战

## 1.Tomcat+Nginx集群架构概述

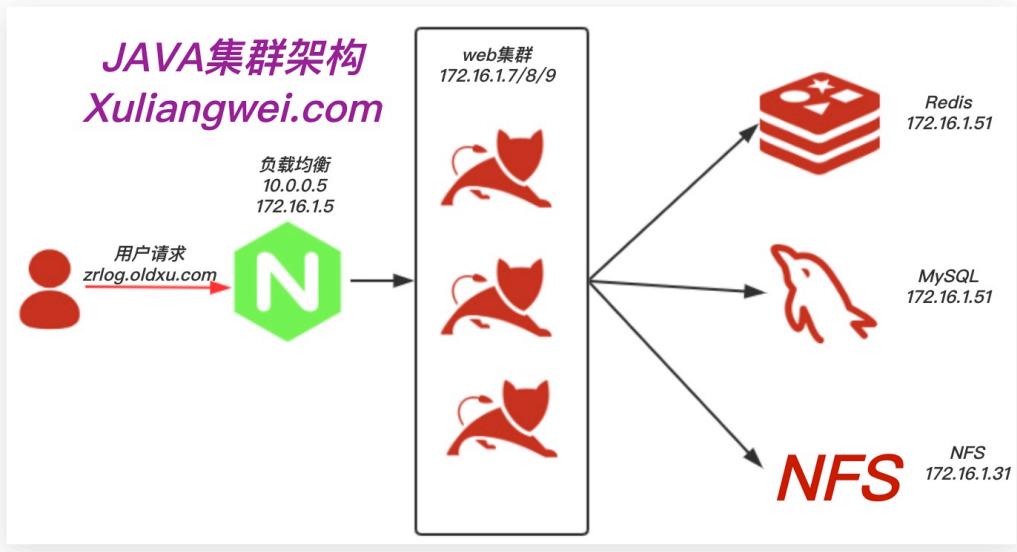
### 1.Tomcat集群能带来什么

1. 提高服务的性能、并发能力、以及高可用性
2. 提高项目架构的扩展能力

### 2.Tomcat集群实现原理

通过Nginx负载均衡进行请求转发

### 3.Tomcat集群架构图



## 2.Tomcat+Nginx集群架构实战

实践环境

服务器系统	角色	外网IP	内网IP
CentOS 7.5	NginxProxy	eth0:10.0.0.5	eth1:172.16.1.5
CentOS 7.5	web-Node1		eth1:172.16.1.7
CentOS 7.5	web-Node2		eth1:172.16.1.8
CentOS 7.5	web-Node3		eth1:172.16.1.9

1.web 节点按如下安装好Tomcat即可

```
[root@web01 ~]# yum install java -y
[root@web01 ~]# mkdir /soft && cd /soft
[root@web01 soft]# wget http://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-9/v9.0.26/bin/apache-tomcat-9.0.26.tar.gz
[root@web01 soft]# tar xf apache-tomcat-9.0.26.tar.gz
[root@web01 soft]# ln -s /soft/apache-tomcat-9.0.26 /soft/tomcat
```

2.安装Nginx，并配置负载均衡

```
[root@lb01 ~]# cat /etc/yum.repos.d/nginx.repo
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/$basearch/
gpgcheck=0
```

```
enabled=1

[root@lb01 ~]# yum install nginx -y
[root@lb01 ~]# cat /etc/nginx/conf.d/proxy.conf.bak
upstream java {
    server 172.16.1.7:8080;
    server 172.16.1.8:8080;
    server 172.16.1.9:8080;
}

server {
    server_name nginx.bjstack.com;
    listen 80;
    location / {
        proxy_pass http://java;
        include proxy_params;
    }
}
```

### 3.准备负载均衡相关优化参数文件

```
[root@lb01 ~]# cat /etc/nginx/proxy_params
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

proxy_connect_timeout 30;
proxy_send_timeout 60;
proxy_read_timeout 60;
```

## 3.Tomcat+Nginx+Https实践

### 1.Tomcat单机配置证书【了解即可】[阿里云文档传送门](#)

```
[root@web01 ~]# vim Tomcat安装目录/conf/server.xml
# <!--http连接器-->处理请求与响应请求-->
<Service name="Catalina">
    <Connector port="80" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="443" />

# <!--https连接器-->
<Connector port="443" #port属性根据实际情况修改 (https默认端口为443)
           protocol="HTTP/1.1"
           SSLEnabled="true"
```

```

scheme="https"
secure="true"
keystoreFile="/ssl/3434295_aliyun.xuliangwei.com.pfx" #证书名称需要添加绝对路径
。
keystoreType="PKCS12"
keystorePass="OpI94943" ##替换为密码文件pxf-password.txt中的内容
clientAuth="false"
SSLProtocol="TLSv1+TLSv1.1+TLSv1.2"
ciphers="TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA256"/>

#2. 配置web.xml文件，开启HTTP强制跳转HTTPS。在文件</welcome-file-list>后添加以下内容：
<login-config>
    <!-- Authorization setting for SSL -->
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>Client Cert Users-only Area</realm-name>
</login-config>
<security-constraint>
    <!-- Authorization setting for SSL -->
    <web-resource-collection >
        <web-resource-name >SSL</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>

```

2.Nginx结合Tomcat配置Https，仅需在Nginx负载均衡配置证书即可。【生产必用】

```

#1. 创建ssl目录，并将证书存放至该目录即可
[root@lb01 ~]# mkdir /etc/nginx/ssl_key && /etc/nginx/ssl_key
[root@lb01 ~]# rz #上传证书

#2. 配置Proxy调度策略以及支持https
[root@lb01 ~]# cat /etc/nginx/conf.d/proxy.conf
upstream java {
    server 172.16.1.7:8080;
    server 172.16.1.8:8080;
    server 172.16.1.9:8080;
}

server {
    server_name tomcat.oldxu.com;
    listen 443;
    ssl on;

```

```

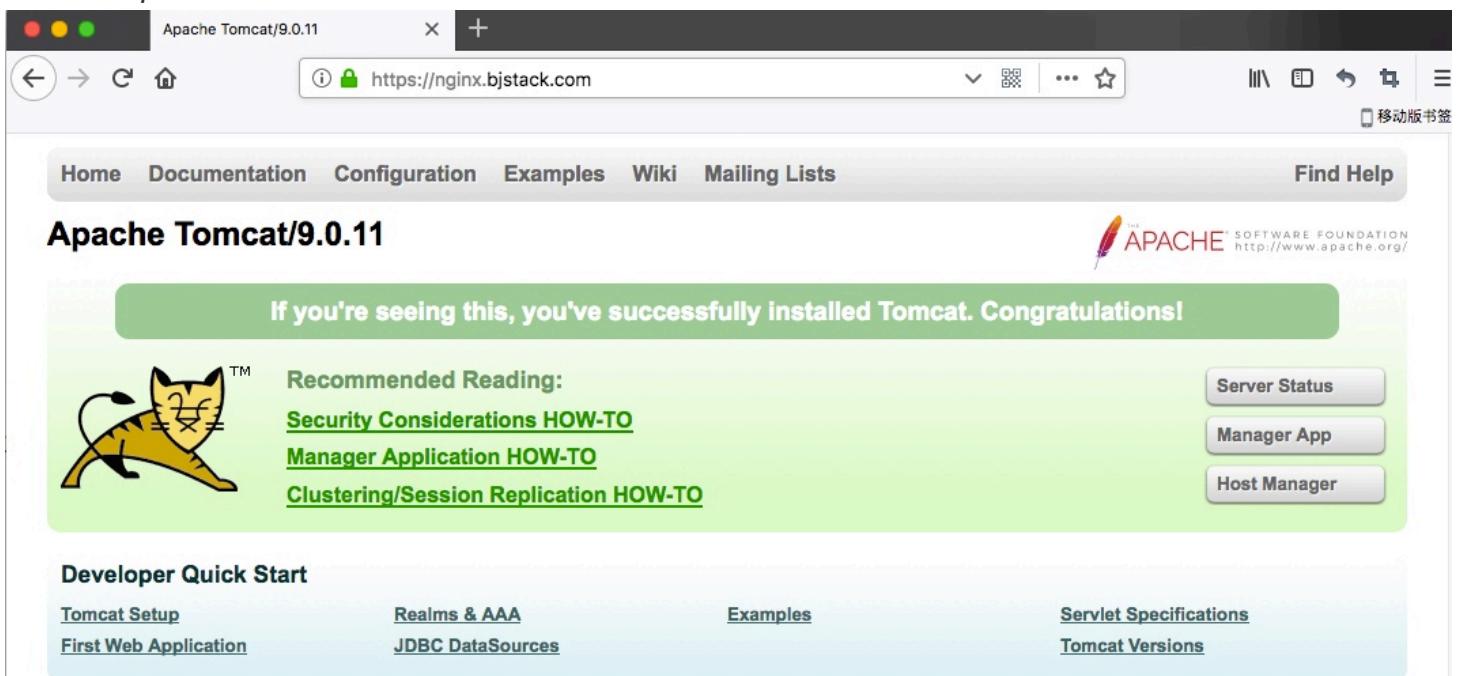
ssl_certificate    ssl/1524377920931.pem;
ssl_certificate_key  ssl/1524377920931.key;
ssl_session_timeout 5m;
location / {
    proxy_pass http://java;
    include proxy_params;
}
}

server {
    listen 80;
    server_name tomcat.oldxu.com;
    return 302 https://$server_name$request_uri;
}

```

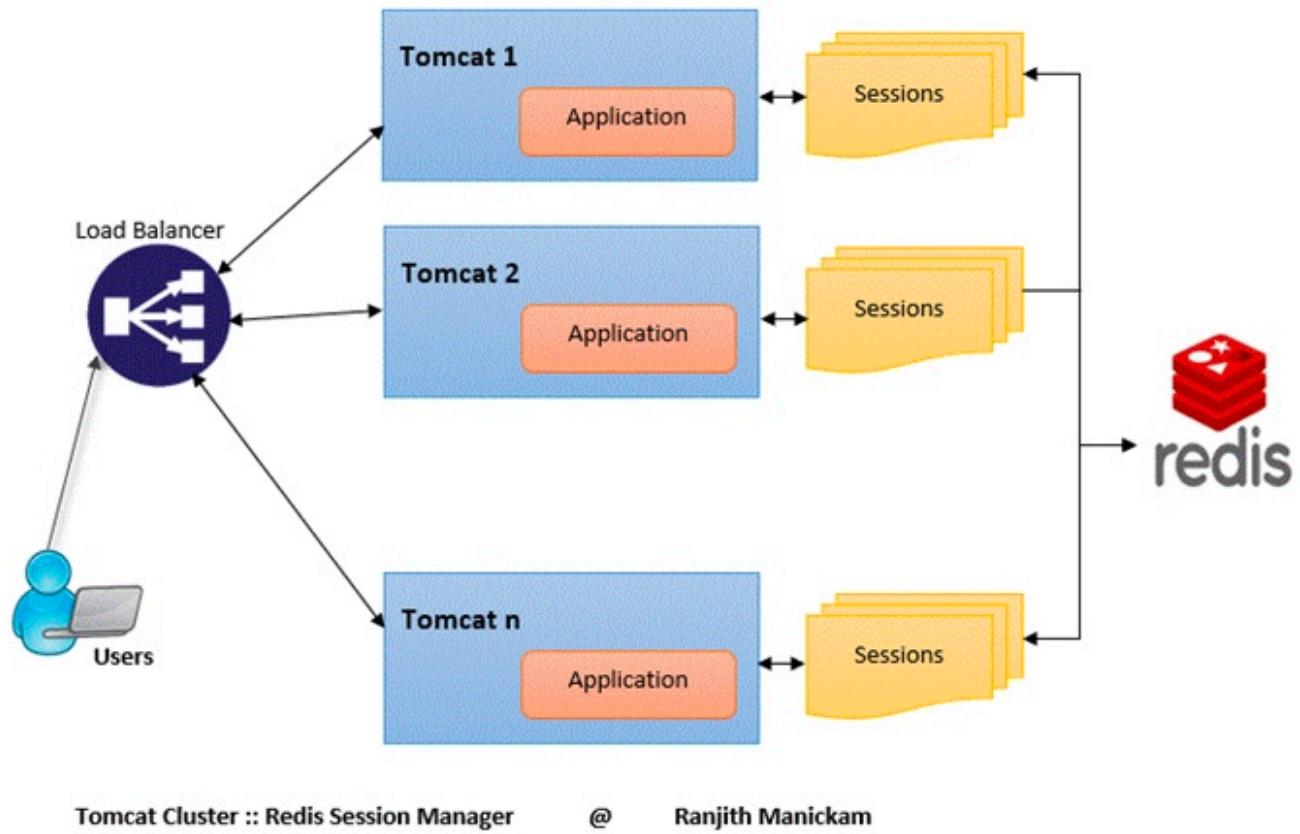
#3. 重启Nginx  
[root@lb01 ~]# systemctl restart nginx

### 3. 通过https访问效果



## 4. Tomcat+Nginx+Redis实践

redis的方式来实现Tomcat的session共享: 架构图如下:



**Tomcat Cluster :: Redis Session Manager** @ Ranjith Manickam

## 主机规划

主机名	IP地址	服务
lb	10.0.0.5,172.16.1.5	nginx
web01	172.16.1.7	Tomcat-9
web02	172.16.1.8	Tomcat-9
redis	172.16.1.51	Redis-3

1. 此前已经安装好Nginx负载均衡+Tomcat集群，所以仅需在所有web节点新增一个虚拟主机。

```
[root@tomcat ~]# vim /soft/tomcat/conf/server.xml
<!--session站点-->
<Host name="session.oldxu.com" appBase="/code/session"
      unpackWARs="true" autoDeploy="true">
</Host>
```

2. 添加一个测试session的jsp页面

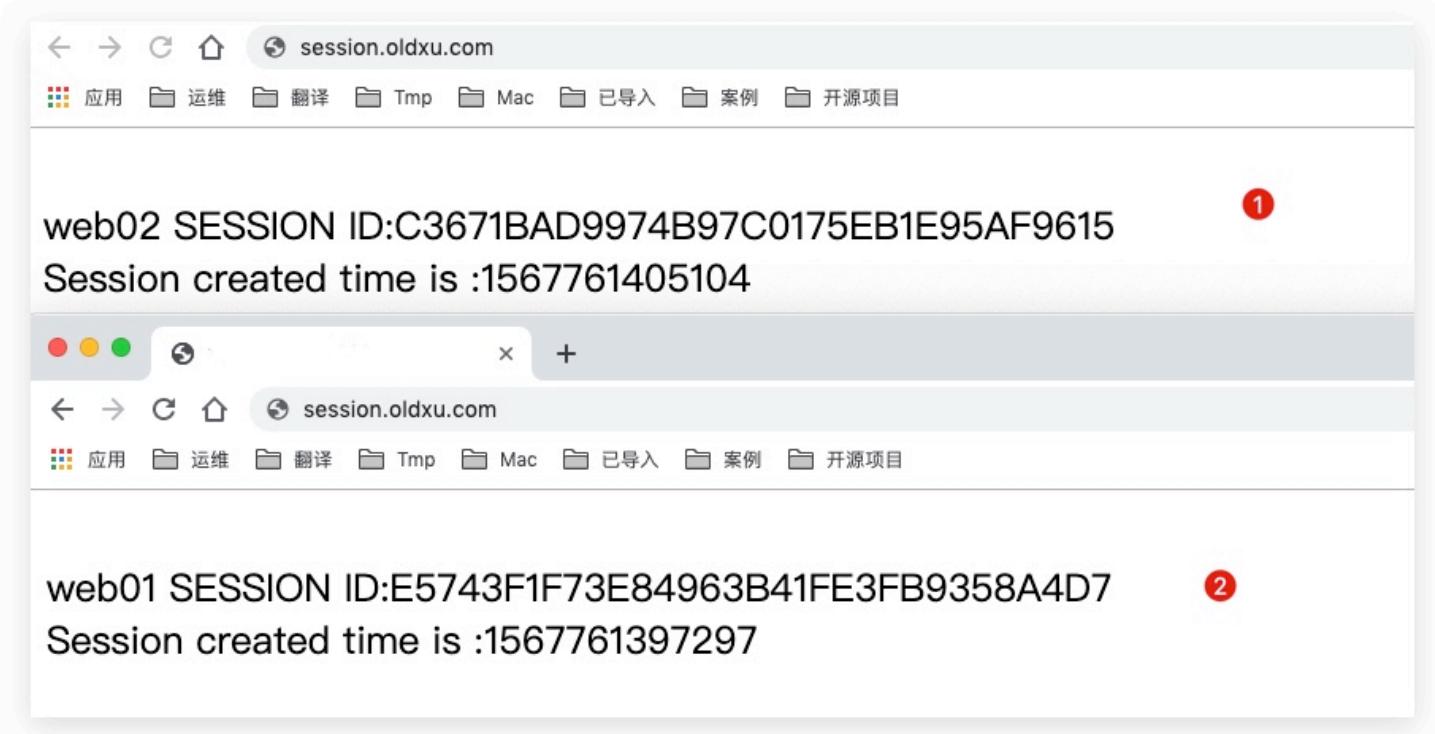
```
[root@tomcat ~]# cat /code/session/ROOT/index.jsp
```

```
<body>
<%
//HttpSession session = request.getSession(true);
System.out.println(session.getCreationTime());
out.println("<br> web01 SESSION ID:" + session.getId() + "<br>");
out.println("Session created time is :" + session.getCreationTime()
+ "<br>");
%>
</body>
```

### 3. 重启tomcat，然后配置域名劫持。

```
[root@tomcat ~]# /soft/tomcat/bin/shutdown.sh && /soft/tomcat/bin/startup.sh && tail -f /soft/tomcat/logs/catalina.out
```

### 4. 测试访问，通过访问nginx，然后轮询调度到后端web集群，会发现不同主机的session不一致。



## 配置TomcatClusterRedisSessionManager方式实现redis共享(此方式支持redis集群)

### 1. 安装redis，当然也可以自行搭建redis集群，anyway

```
[root@redis ~]# yum install redis -y
[root@redis ~]# cat /etc/redis.conf
...
bind 172.16.1.51 127.0.0.1
requirepass 123456
```

```
...
[root@redis ~]# systemctl start redis
[root@redis ~]# systemctl enable redis
```

## 2. 下载TomcatClusterRedisSessionManager (所有web集群都需要操作) [GitHub地址](#)

```
[root@tomcat ~]# wget https://github.com/ran-jit/tomcat-cluster-redis-session-manager/releases/download/3.0.3/tomcat-cluster-redis-session-manager.zip
[root@tomcat ~]# unzip tomcat-cluster-redis-session-manager.zip
```

## 3. 将下载的jar移动到tomcat/lib目录

```
[root@tomcat ~]# cp tomcat-cluster-redis-session-manager/lib/* /soft/tomcat/lib/
```

## 4. 将redis-data-cache.properties文件移动到tomcat/conf目录，并完成配置

```
[root@tomcat ~]# cp tomcat-cluster-redis-session-manager/conf/redis-data-cache.properties /soft/tomcat/conf/
[root@tomcat ~]# vim /soft/tomcat/conf/redis-data-cache.properties
...
redis.hosts=172.16.1.51:6379
redis.password=123456
redis.cluster.enabled=false
...
```

## 5. 在tomcat/conf/context.xml中添加以下两行

```
<Valve className="tomcat.request.session.redis.SessionHandlerValve" />
<Manager className="tomcat.request.session.redis.SessionManager" />
```

## 6. 验证tomcat/conf/web.xml中的会话过期时间（分钟）

```
<session-config>
    <session-timeout>60</session-timeout>
<session-config>
```

## 7. 最后启动所有的tomcat

```
[root@tomcat ~]# /soft/tomcat/bin/startup.sh
```

8. 通过nginx 进行访问，每次刷新SessionID都是一致的，说明基于Redis的session共享方式部署成功。

The screenshot displays two browser windows and a terminal window. The top browser window shows the session ID 'SESSION ID:FD0CD98F056584DA56EA37A9C8E07883' highlighted in red. The bottom browser window shows the same session ID. To the right, a terminal window shows Redis commands and responses related to the session key.

Terminal Output:

```
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> keys *
1) "FD0CD98F056584DA56EA37A9C8E07883"
127.0.0.1:6379>
```

Session Information:

session 信息已存储至 redis 中