# 基于协议和配置的HTTPS优化方案

## 1.1 https所带来的额外开销

对于用户来说，使用HTTP请求，首次请求时要和服务端TCP三次握手建立连接，就能保证用户直接通过HTTP请求获取应用层数据。同时，keep-alive机制能够保证客户端与服务端的连接持续可用。

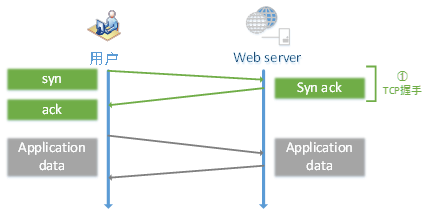


图7. 用户的HTTP请求

HTTPS会给用户带来更多的额外开销：

（1）302跳转,，1个RTT时间。用户输入http://www.suning.com，服务器返回302跳转的状态码，跳转到https://www.suning.com。注意，浏览器处理302跳转也需要耗时。

（2）TLS完全握手阶段1，证书身份认证，1个RTT时间。如果浏览器中标注证书状态的缓存过期，还需要到CA去验证证书的合法性，所带来的开销就更大了。

（3）TLS完全握手阶段2，密钥协商过程，1个RTT时间。该阶段结束后，浏览器与服务器之间进行应用层数据传输。

（4）消耗CPU资源的计算耗时。以上只考虑HTTPS传输过程中额外消耗的纯网络耗时，还需考虑CPU计算耗时（百度HTTPS经验是手机端纯计算延迟50ms以上，PC端10ms以上）。主要包括4个方面：a. 证书签名校验，b. 根据公钥计算key值，c. 对称加解密，d. 一致性校验。

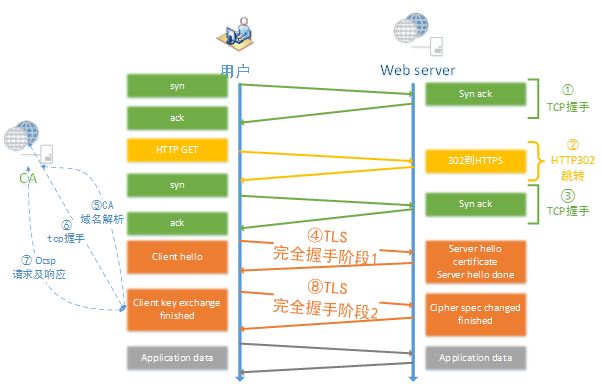


图8. 用户的HTTPS请求

## 1.2 HSTS的合理使用

### 1.2.1 如何避免302跳转带来的性能开销？

这个问题可以通过HSTS（HTTP Strict Transport Security）来解决，当浏览器第一次HTTP请求服务器时，返回的响应头中带一个标签Strict-Transport-Security，告诉浏览器在指定的时间内，这个网站必须通过HTTPS协议来访问。也就是对于这个网站的HTTP地址，浏览器需要现在本地替换为HTTPS之后再发送请求。

|  |
| --- |
| Strict-Transport-Security：max-age=expireTime [; includeSubDomains] [; preload] |

目前，主流浏览器对HSTS都是支持的（IE只有IE 11支持）。

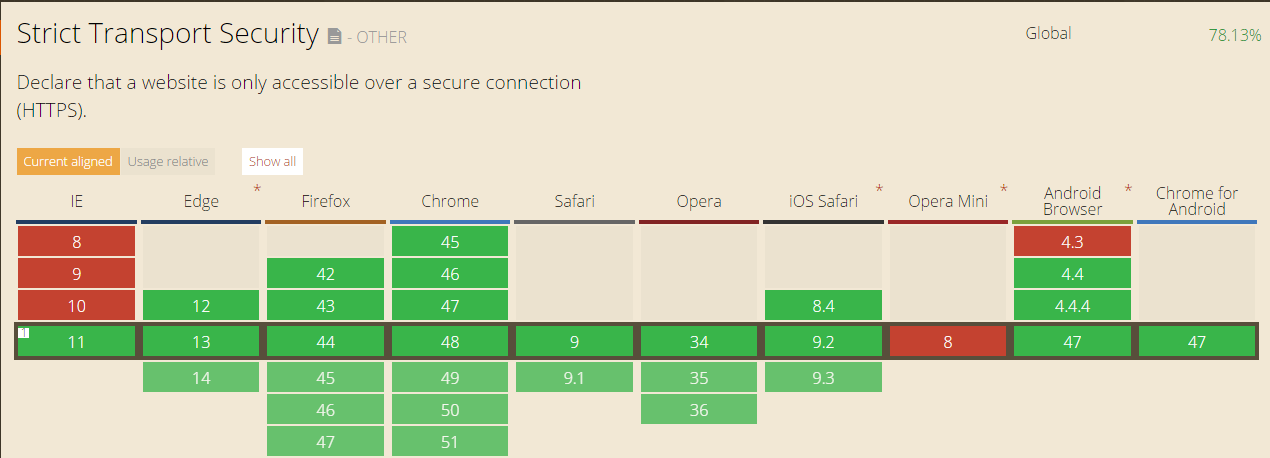


图9. 支持HSTS的浏览器

### 1.2.2 HSTS的利弊

优势：减少了HTTP做302跳转的开销。302跳转不仅暴露了用户的访问站点，也很容易被中间者支持（降级劫持、中间人攻击），最重要是降低了访问速度（影响性能）。

缺点：

1. HSTS在max-age过期时间内在客户端是强制HTTPS的，服务端无法控制。因此，需要降级时，HTTPS无法及时切换到HTTP。

2. HSTS是严格的HTTPS，一旦网络证书错误时，网页将直接无法访问（用户无法选择忽视）。



图10. 使用了HSTS后，证书错误将无法选择忽视

### 1.2.3 各家公司使用HSTS情况

在是否使用HSTS上，各家是不同的。目前，只有淘宝使用了HSTS，并缓存一个月的时间。京东、百度、谷歌等仍没有使用HSTS，而是直接301、302跳转。

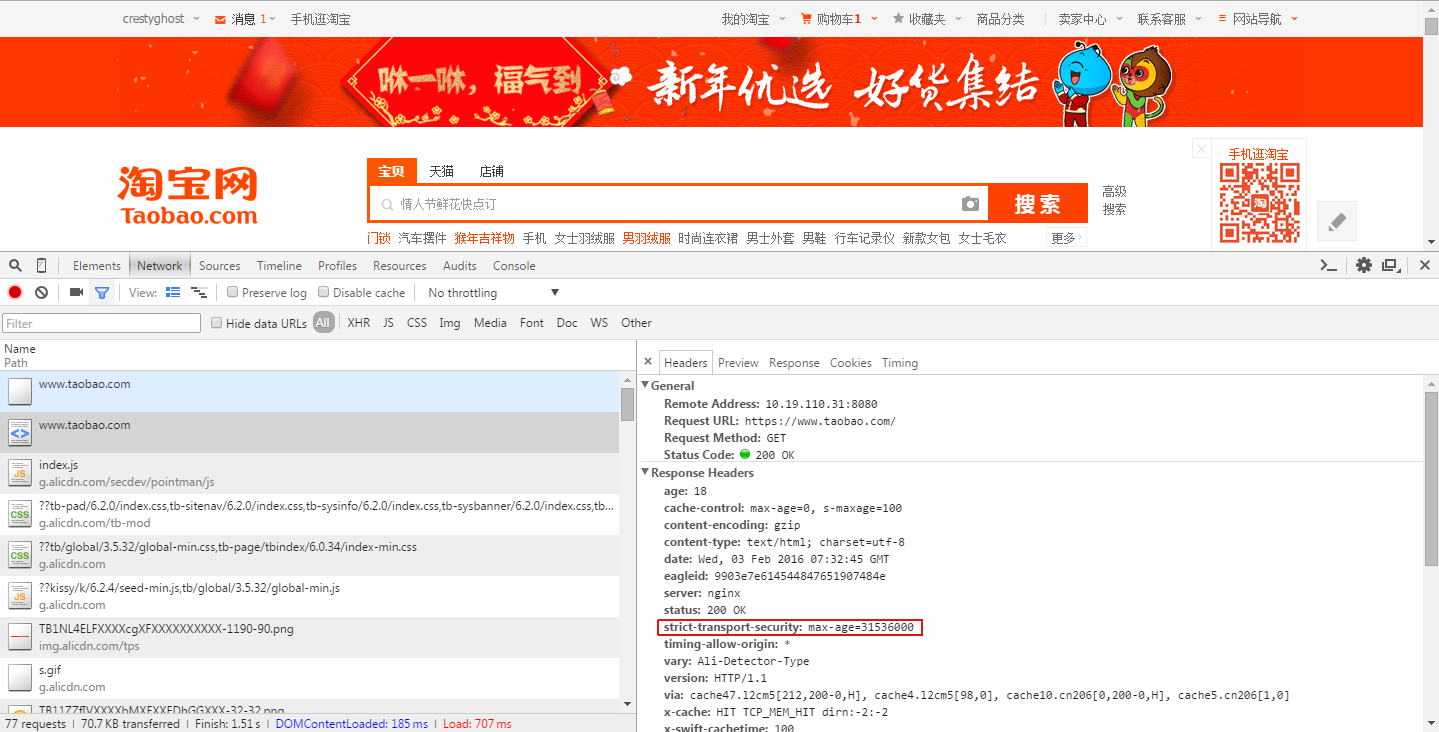


图11. 淘宝使用了HSTS，缓存一个月左右

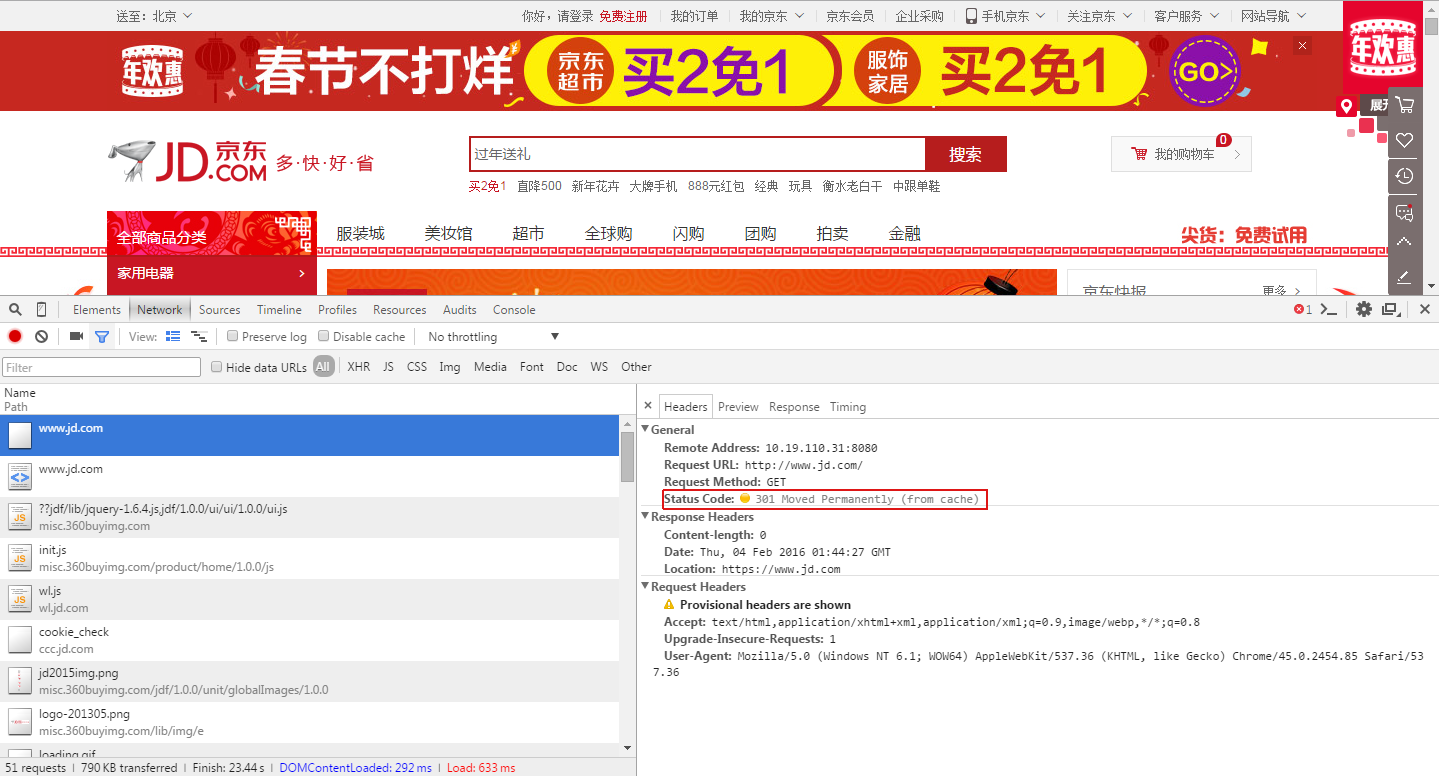


图12 京东没有使用HSTS，直接301跳转

### 1.2.4 如何在服务器上配置HSTS

目前，可以在Apache2、Nginx和Lighttpd上启用HSTS。

（1）Apache2配置

在apache配置文件httpd.conf中配置，并加以下行到你的HTTPS virtualHost：

|  |
| --- |
| # Optionally load the headers module:  LoadModule headers\_module modules/mod\_headers.so  <VirtualHost 67.89.123.45:443>  Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"  </VirtualHost> |

注意，你只能在HTTPS虚拟机中设置这个头，而不能设置在HTTP虚拟机中。因此，要将你的访问者重定向到对应HTTPS的站点，如下配置

|  |
| --- |
| <VirtualHost \*:80>  [...]  ServerName example.com  Redirect permanent / https://example.com/  </VirtualHost> |

（2）Nginx配置

Nginx配置很简单，只需要在HTTPS对应的server模块中增加（不要忘记之后重启Nginx）：

|  |
| --- |
| add\_header Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"; |

## 1.3 Session resume的合理使用

Session resume，即复用session，实现简化握手。Session resume的优点是：

（1）减少CPU消耗，复用会话使同一客户端之后的请求不需要进行密钥交换的计算；

（2）提升访问速度，避免了完全握手阶段二，节省了一个RTT时间和计算耗时。

目前，TLS协议提供了两种机制实现Session resume，即Session ID和Session ticket。

### 1.3.1 Session ID（RFC 5246）

Session ID的原理：客户端和服务器端都保存了会话key，通过每次连接给出的唯一标识Session ID，服务器知道这个连接之前是否被创建过（即会话key是否已经在服务器端），如果被创建过，它就能重用。

#### 4.3.1.1 Session ID的优缺点：

优点：兼容性好，市面上的浏览器都支持Session ID；

缺点：（1）保存会话key消耗服务器端内存；（2）只支持单机但进程共享缓存，不支持分布式集群环境。

注：参照Nginx的官方文档1MB内存大约可以存储4000个session，按例配置20M大约可以存储80000。根据需求合理设置

但经过研究，Session ID的缺点都是可以解决的。Nginx+lua+memcache



图13. Nginx+lua+memcache解决Session ID不支持集群的问题

#### 1.3.1.2 如何在Nginx上使用Session ID实现会话重用

在Nginx HTTPS的server模块中配置：

|  |
| --- |
| ssl\_protocols TLSv1 TLSv1.1 TLSv1.2;  ssl\_ciphers ECDHE-RSA-AES256-SHA384:AES256-SHA256:RC4:HIGH:!MD5:!aNULL:!eNULL:!NULL:!DH:!EDH:!AESGCM;  ssl\_prefer\_server\_ciphers on;  ssl\_session\_cache shared:SSL:10m;  ssl\_session\_timeout 10m; |

（1） Nginx默认设置的DH算法（译注：Diffie-Hellman key exchange algorithm）是影响SSL性能的最大因素，因此采用如上设置能增加SSL性能

（2） 升级硬件 Upgrade your EC2 from t1.micro to c1.medium

（3） 正确配置 Nginx的worker进程数量 Set Nginx to have 2 worker processes as a c1.medium gives you 2 CPUs

### 1.3.2 Session tickets（RFC 5077）

Session ticket的原理：ticket是一个加密的数据段，包含需要重用的TLS连接信息（会话key），使用ticket key加解密（因此服务器端也知道）。服务器端将ticket发送给浏览器，以后浏览器所有后续的握手请求都使用ticket，完成简化握手，会话重用。

#### 1.3.2.1 Session ticket的优缺点：

优点：不需要服务器端消耗大量内存存储session内容

缺点：（1）兼容性不好，目前浏览器的支持率不到60%

（2）session ticket需要维护一个全局的key来加密，要考虑key的安全性和部署效率（当集群中增加一台新的nginx，需要停机部署！）

问题（2）被CloudFlare 和 Twitter使用系统产生一个集中统一的key来解决，ticket key被一个集中的统一的服务器定期创建，安全地发给所有服务器，实现会话ticket共享需要你的架构有一个定制系统的抉择。（参考文章较少）

#### 1.3.2.2 如何在Nginx上使用Session ticket实现会话重用？

参考文章较少，第一步是开启nginx的ssl\_session\_tickets，然后自己写lua脚本实现。

|  |
| --- |
| ssl\_session\_tickets on;  ssl\_session\_ticket\_key \*/ticket.key; |

综上，2.3.1和2.3.2，建议使用Session ID的方式实现SSL会话的重用，提高HTTPS访问效率。

## 1.4 Ocsp stapling的合理使用（证书链优化）

Ocsp，即在线证书状态检查协议（RFC6960），用来向CA站点查询证书状态，比如是否撤销，通常情况下，浏览器使用OSCP协议发起查询请求，CA返回证书状态内容，然后浏览器接受证书是否可信的状态。

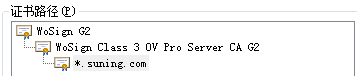


图14. 目前苏宁使用沃通的证书链

这个过程非常消耗性能，因为CA站点有可能在国外，网络不稳定，RTT也比较大。

### 1.4.1 Ocsp stapling的优点

Ocsp stapling由服务器代替浏览器向CA站点查询证书状态，这个功能对访问速度提升非常明显。

### 1.4.2 如何在Nginx上实现Ocsp stapling

Nginx目前已经支持了Ocsp stapling file，只需要配置指令就能开启功能：

|  |
| --- |
| ssl\_stapling on;  ssl\_stapling\_verify on;  ssl\_stapling\_file /stapling\_file.ocsp; //证书状态验证文件  ssl\_trusted\_certificate /certificate-path/trustchain.crt; //适用证书 |

需要注意的是ocsp证书有效期很短，大概一个月不到，所以要写lua脚本定期更新ocsp证书。

## 1.5 启用SPDY/HTTP2

SPDY 最大的特性就是多路复用，能将多个 HTTP 请求在同一个连接上一起发出去，不像目前的 HTTP 协议一样，只能串行地逐个发送请求。Pipeline（keep-alive）虽然支持多个请求一起发送，但是接收时依然得按照顺序接收，本质上无法解决并发的问题。

需要说明两点：

（1）SPDY和HTTP2 目前的实现默认使用 HTTPS 协议。

（2）SPDY和HTTP2 都支持现有的HTTP语义和API，对WEB应用几乎是透明的。

在Nginx上带上参数--with-http\_spdy\_module支持SPDY协议，并在配置中启用：

|  |
| --- |
| listen 443 ssl spdy; |

## 1.6 加密/协议安全设置

### 1.6.1 加密安全设置

（1）非对称密钥交换算法。建议优先使用 ECDHE，禁用 DHE，次优先选择 RSA；

（2）证书签名算法。由于部分浏览器及操作系统不支持 ECDSA 签名，目前默认都是使用 RSA 签名，其中 SHA1 签名已经不再安全，chrome 及微软 2016 年开始不再支持 SHA1 签名的证书；

（3）对称加解密算法。优先使用 AES-GCM 算法，针对 1.0 以上协议禁用 RC4（rfc7465）；

（4内容一致性校验算法。Md5 和 sha1 都已经不安全，建议使用 sha2 以上的安全哈希函数。

|  |
| --- |
| ssl\_prefer\_server\_ciphers On; *#指定服务器密码算法在优先于客户端密码算法时，使用SSLv3和TLS协议。*  ssl\_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-RC4-SHA:!ECDHE-RSA-RC4-SHA:ECDH-ECDSA-RC4-SHA:ECDH-RSA-RC4-SHA:ECDHE-RSA-AES256-SHA:!RC4-SHA:HIGH:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!CBC:!EDH:!kEDH:!PSK:!SRP:!kECDH; |

### 1.6.2 协议安全设置

在 SSL 协议这里默认支持的是 SSLv3, TLSv1, TLSv1.1, TLSv1.2（需要 OpenSSL 支持，针对近两三年的 Nginx 版本），因为 SSLv3 的一些安全问题，还有 TLSv1 的浏览器支持基本没有问题了，所以可以直接禁用掉，（当然 TLSv1 也有安全问题，但是 TLSv1.1 支持有限，所以这里暂时先保留着）。

|  |
| --- |
| ssl\_protocols TLSv1 TLSv1.1 TLSv1.2; |

## 1.7 使用最新的openssl

一般来讲，新版的openssl相比老版的计算速度和安全性都会有提升。比如openssl1.0.2采用了intel最新的优化成果，椭圆曲线 p256 的计算性能提升了4倍。

Openssl 2014年就升级了5次，基本都是为了修复实现上的BUG或者算法上的漏洞而升级的。所以尽量使用最新版本，避免安全上的风险。

## 1.8 硬件加速方案

现在比较常用的 TLS 硬件加速方案主要有两种：

（1）SSL 专用加速卡。

（2）GPU SSL 加速。

### 1.8.1 硬件加速的缺点

上述两个方案的主流用法都是将硬件插入到服务器的 PCI 插槽中，由硬件完成最消耗性能的计算。但这样的方案有如下缺点：

（1）支持算法有限。比如不支持 ECC，不支持 GCM 等。

（2）升级成本高。

a. 出现新的加密算法或者协议时，硬件加速方案无法及时升级。

b. 出现比较大的安全漏洞时，部分硬件方案在无法在短期内升级解决。比如 2014 年暴露的 heartbleed 漏洞。

（3）无法充分利用硬件加速性能。硬件加速程序一般都运行在内核态，计算结果传递到应用层需要 IO 和内存拷贝开销，即使硬件计算性能非常好，上层的同步等待和 IO 开销也会导致整体性能达不到预期，无法充分利用硬件加速卡的计算能力。

（4）维护性差。硬件驱动及应用层 API 大部分是由安全厂家提供，出现问题后还需要厂家跟进。用户无法掌握核心代码，比较被动。不像开源的 openssl，不管算法还是协议，用户都能掌握。

### 1.8.2 百度专用SSL 硬件加速集群的案例

百度实现了专用的 SSL 硬件加速集群。基本思路是：

（1）优化 TLS 协议栈，剥离最消耗 CPU 资源的计算，主要有如下部分：

a. RSA 中的加解密计算。

b. ECC 算法中的公私钥生成。

c. ECC 算法中的共享密钥生成。

（2）优化硬件计算部分。硬件计算不涉及协议及状态交互，只需要处理大数运算。

（3）Web server 到 TLS 计算集群之间的任务是异步的。即 web server 将待计算内容发送给加速集群后，依然可以继续处理其他请求，整个过程是异步非阻塞的。