Atitit.HTTP 代理原理及实现 正向代理与反向代理attilax总结

[1. 普通代理 1](#_Toc15548)

[1.1.1. 普通代理 2](#_Toc621)

[2. 隧道代理 3](#_Toc9326)

[3. 反向代理 4](#_Toc13602)

[4. 正向代理也可以使用apache实现 4](#_Toc28435)

[5. 参考 5](#_Toc19592)

HTTP 代理原理及实现（一）

****文章目录****

# [普通代理](https://imququ.com/post/web-proxy.html" \l "toc-0)

Web 代理是一种存在于网络中间的实体，提供各式各样的功能。现代网络系统中，Web 代理无处不在。我之前有关 HTTP 的博文中，多次提到了代理对 HTTP 请求及响应的影响。今天这篇文章，我打算谈谈 HTTP 代理本身的一些原理，以及如何用 Node.js 快速实现代理。

HTTP 代理存在两种形式，分别简单介绍如下：

第一种是 [RFC 7230 - HTTP/1.1: Message Syntax and Routing](http://tools.ietf.org/html/rfc7230)（即修订后的 RFC 2616，HTTP/1.1 协议的第一部分）描述的普通代理。这种代理扮演的是「中间人」角色，对于连接到它的客户端来说，它是服务端；对于要连接的服务端来说，它是客户端。它就负责在两端之间来回传送 HTTP 报文。

第二种是 [Tunneling TCP based protocols through Web proxy servers](https://tools.ietf.org/html/draft-luotonen-web-proxy-tunneling-01)（通过 Web 代理服务器用隧道方式传输基于 TCP 的协议）描述的隧道代理。它通过 HTTP 协议正文部分（Body）完成通讯，以 HTTP 的方式实现任意基于 TCP 的应用层协议代理。这种代理使用 HTTP 的 CONNECT 方法建立连接，但 CONNECT 最开始并不是 RFC 2616 - HTTP/1.1 的一部分，直到 2014 年发布的 HTTP/1.1 修订版中，才增加了对 CONNECT 及隧道代理的描述，详见 [RFC 7231 - HTTP/1.1: Semantics and Content](https://tools.ietf.org/html/rfc7231" \l "section-4.3.6)。实际上这种代理早就被广泛实现。

本文描述的第一种代理，对应《HTTP 权威指南》一书中第六章「代理」；第二种代理，对应第八章「集成点：网关、隧道及中继」中的 8.5 小节「隧道」。

.作者:: 绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ） 汉字名：艾龙，  EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax

### 普通代理

第一种 Web 代理原理特别简单：

HTTP 客户端向代理发送请求报文，代理服务器需要正确地处理请求和连接（例如正确处理 Connection: keep-alive），同时向服务器发送请求，并将收到的响应转发给客户端。

当然代理也可以修改 HTTP 请求头部，通过 X-Forwarded-IP 这样的自定义头部告诉服务端真正的客户端 IP。但服务器无法验证这个自定义头部真的是由代理添加，还是客户端修改了请求头，所以从 HTTP 头部字段获取 IP 时，需要格外小心。

# [隧道代理](https://imququ.com/post/web-proxy.html" \l "toc-1)

可以看到，浏览器与代理进行 TCP 握手之后，发起了 CONNECT 请求，报文起始行如下：

CONNECT imququ.com:443 HTTP/1.1

对于 CONNECT 请求来说，只是用来让代理创建 TCP 连接，所以只需要提供服务器域名及端口即可，并不需要具体的资源路径。代理收到这样的请求后，需要与服务端建立 TCP 连接，并响应给浏览器这样一个 HTTP 报文：

HTTP/1.1 200 Connection Established

浏览器收到了这个响应报文，就可以认为到服务端的 TCP 连接已经打通，后续直接往这个 TCP 连接写协议数据即可。通过 Wireshark 的 Follow TCP Steam 功能，可以清楚地看到浏览器和代理之间的数据传递：

可以看到，浏览器建立到服务端 TCP 连接产生的 HTTP 往返，完全是明文，这也是为什么 CONNECT 请求只需要提供域名和端口：如果发送了完整 URL、Cookie 等信息，会被中间人一览无余，降低了 HTTPS 的安全性。HTTP 代理承载的 HTTPS 流量，应用数据要等到 TLS 握手成功之后通过 Application Data 协议传输，中间节点无法得知用于流量加密的 master-secret，无法解密数据。而 CONNECT 暴露的域名和端口，对于普通的 HTTPS 请求来说，中间人一样可以拿到（IP 和端口很容易拿到，请求的域名可以通过 DNS Query 或者 TLS Client Hello 中的 Server Name Indication 拿到），所以这种方式并没有增加不安全性

# 反向代理

还有一种情况是访问 A 网站时，实际上访问的是代理，代理收到请求报文后，再向真正提供服务的服务器发起请求，并将响应转发给浏览器。这种情况一般被称之为反向代理，它可以用来隐藏服务器 IP 及端口。一般使用反向代理后，需要通过修改 DNS 让域名解析到代理服务器 IP，这时浏览器无法察觉到真正服务器的存在，当然也就不需要修改配置了。反向代理是 Web 系统最为常见的一种部署方式，例如本博客就是使用 Nginx 的 proxy\_pass 功能将浏览器请求转发到背后的 Node.js 服务。

通常是由apache实现

# 正向代理也可以使用apache实现

#正向代理设置

ProxyRequests On

ProxyVia On

<Proxy \*>

Order deny,allow

Deny from all

Allow from 127.0.0.1

</Proxy></VirtualHost>

现在看正向代理设置那一段

* ProxyRequests On：开启Apache正向代理
* ProxyVia On：控制位于代理服务器链中的代理请求的流向

　　引用Apache2.2官方文档中对ProxyVia的解释如下：

* 1. 如果设置为默认值Off ，将不会采取特殊的处理。如果一个请求或应答包含"Via:"头，将不进行任何修改而直接通过。
  2. 如果设置为On每个请求和应答都会对应当前主机得到一个"Via:"头。
  3. 如果设置为Full ，每个产生的"Via:"头中都会额外加入Apache服务器的版本，以"Via:"注释域出现。
  4. 如果设置为Block ，每个代理请求中的所有"Via:"头行都将被删除。且不会产生新的"Via:"头。

# 参考

HTTP 代理原理及实现（一） JerryQu 的小站.htm

Apache配置正向代理与反向代理 - Alexis\_Liu - 博客园.htm