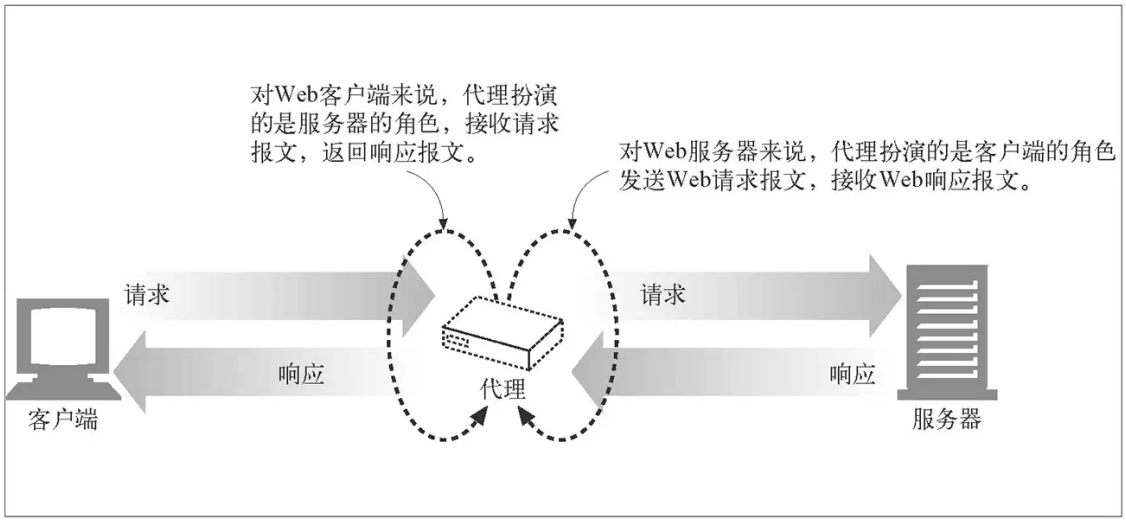
先问一个问题：你的代码中是如何获取客户端真实IP的？本文我想着重与你探讨下，尤其是遇到存在代理的情况。

我们平时会遇到很多中介，尤其是在买二手房时，在本文，我将中介分为两种具体角色：代理和网关，他们很像，有的时候就说成了一样东西，但实际上是有区别的，我们必须弄清楚。

当然了，这两个概念区分还是比较简单的，不足以用一篇宝贵的文章来浪费时间，本文还结合如何在存在反向代理的情况下获取客户端真实IP的实际问题，进行代理的深入探究，让我们出发！

一、HTTP中介之代理

所谓的“代理服务”就是指服务本身不生产内容，而是处于中间位置转发上下游的请求和响应，具有双重身份：面向下游的用户时，表现为服务器，代表源服务器响应客户端的请求；而面向上游的源服务器时，又表现为客户端，代表客户端发送请求。

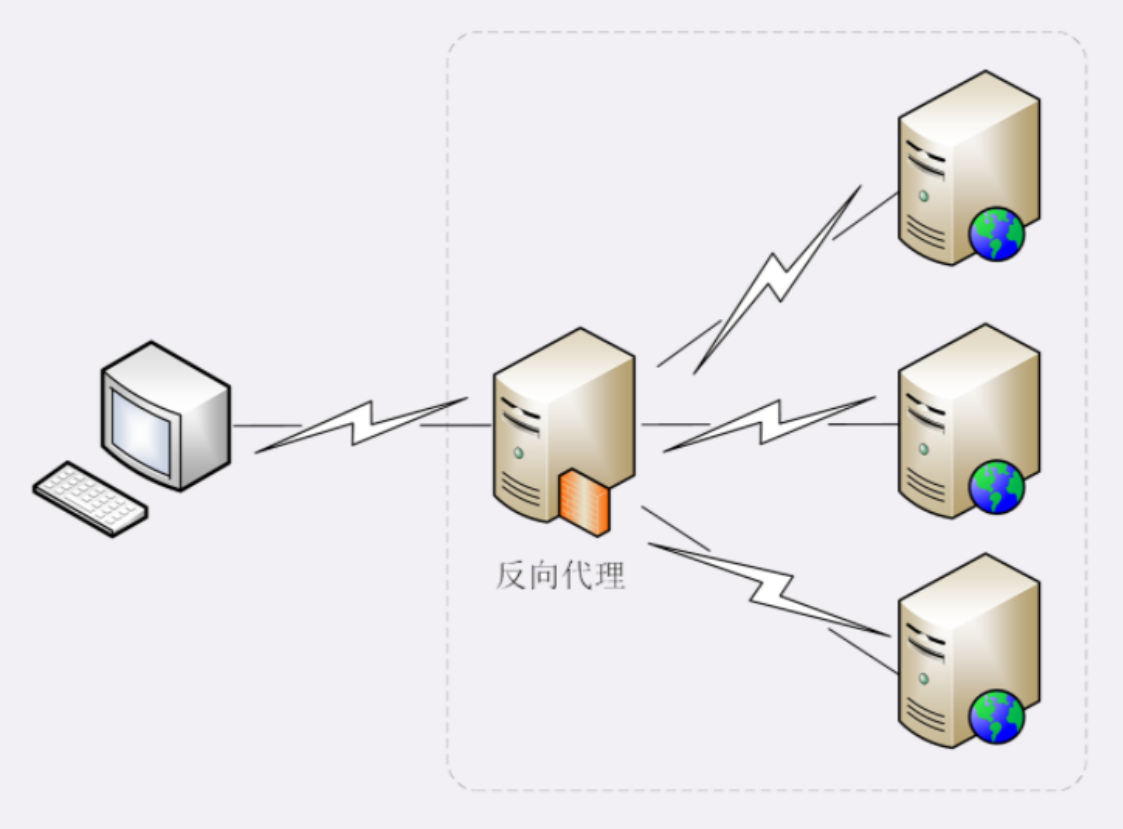


这很好理解，打个比方，之前你都是从超市里买东西，现在楼底下新开了一家 24 小时便利店，由超市直接供货，于是你就可以在便利店里买到原本必须去超市才能买到的商品。这样超市就不直接和你打交道了，成了“源服务器”，便利店就成了超市的“代理服务器”。

为什么要有代理呢？换句话说，代理能干什么、带来什么好处呢？

在计算机科学领域里的任何问题，都可以通过引入一个中间层来解决，不行就再加一层，我们的TCP/IP五层模型或者OSI七层模型正是体现了此思想。

代理到底能解决什么问题呢？代理最常用的一个功能是负载均衡，因为在面向客户端时屏蔽了源服务器，客户端看到的只是代理服务器，源服务器究竟有多少台、是哪些 IP 地址都不知道。于是代理服务器就可以掌握请求分发的“大权”，决定由后面的哪台服务器来响应请求。这就是大名鼎鼎的**反向代理**。



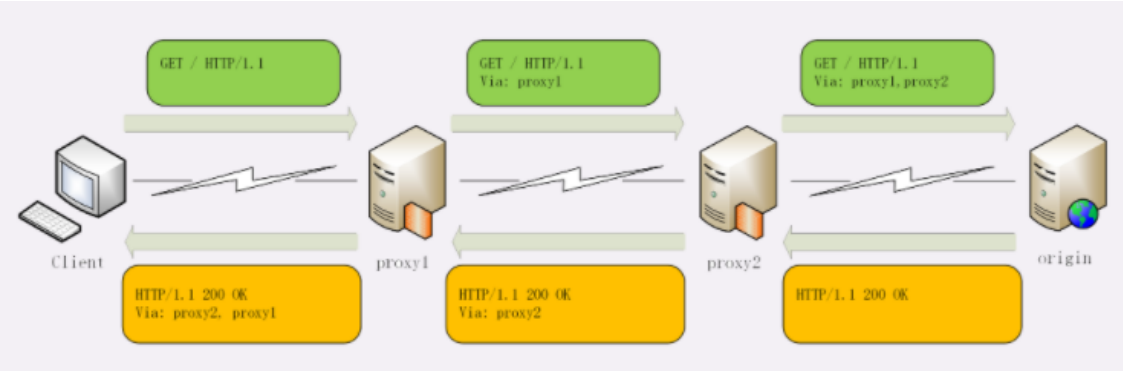
由于代理处在 HTTP 通信过程的中间位置，相应地就对上屏蔽了真实客户端，对下屏蔽了真实服务器，简单的说就是“欺上瞒下”。在这个中间层的“小天地”里就可以做很多的事情，为 HTTP 协议增加更多的灵活性，实现客户端和服务器的“双赢”。

因为它“欺上瞒下”的特点，隐藏了真实客户端和服务器，如果双方想要获得这些“丢失”的原始信息，该怎么办呢?

首先，我们如何知道通信链路中是否经过了中间代理呢?

可以通过 Via 这个通用字段，请求头或响应头里都可以出现。每当报文经过一个代理节点，代理服务器就会把自身的信息追加到字段的末尾，就像是经手人盖了一个章。

例如下图中有两个代理：proxy1 和 proxy2，客户端发送请求会经过这两个代理，依次添加就是 Via: proxy1, proxy2 ，等到服务器返回响应报文的时候就要反过来走，头字段就是 Via: proxy2, proxy1 。



Via 只是解决了如何判断是否有代理，如果我想获取原始客户端的真实IP呢?

二、X-Forwarded-For和X-Real-IP

X-Forwarded-For和X-Real-IP只有请求存在代理时才有值。

我们先来了解一个重要的HTTP头字段：X-Forwarded-For。

X-Forwarded-For 的字面意思是“为谁而转发”，形式上和 Via 差不多，也是每经过一个代理节点就会在字段里追加一个信息。但 Via 追加的是代理主机名（或者域名），而 X-Forwarded-For 追加的是请求方的 IP 地址。所以，在字段里最左边的 IP 地址就客户端的地址。

```
X-Forwarded-For: IP0, IP1, IP2
```

X-Real-IP 是另一种获取客户端真实 IP 的手段，它的作用很简单，就是记录客户端 IP 地址，没有中间的代理信息，相当于是 X-Forwarded-For 的简化版。如果客户端和源服务器之间只有一个代理，那么这两个字段的值就是相同的。

由于HTTP的header可以随意地构造，对于安全要求较高的场景，获取IP需要格外的小心，因为没有代理服务器的情况和有代理服务器的情况，获取IP的方式可能不一样。

如果确定是直连服务的话， request.getRemoteAddr() 获取到的就是用户最真实的IP，为什么这么说呢?

众所周知TCP/IP建立连接时是需要三次握手的，并且，只有知道了client端请求的IP地址，server端的数据才能返回给client，所以client想要获取到数据就必须提供真实的IP。因此 Remote Address 无法伪造，因为建立 TCP 连接需要三次握手，如果伪造了源 IP，无法建立 TCP 连接，更不会有后面的 HTTP 请求。因此说， request.getRemoteAddr() 获取到的就是用户最真实的IP。在没有任何反向代理的情况下，这个方式是可行的。

但是，现在大多数据服务器为了安全都会使用nginx作为代理服务器，而用户对代理服务器发起的HTTP请求，代理服务器对服务集群中的真实部署的对应服务进行“二次请求”，所以最终获取的IP是代理服务器在内网中的ip地址，如 192.168.xx.xx/10.xx.xx.xx 等等。

好了，理论说的差不多了，不动手试试，谁知道说的对不对，让我们来动起手来吧。

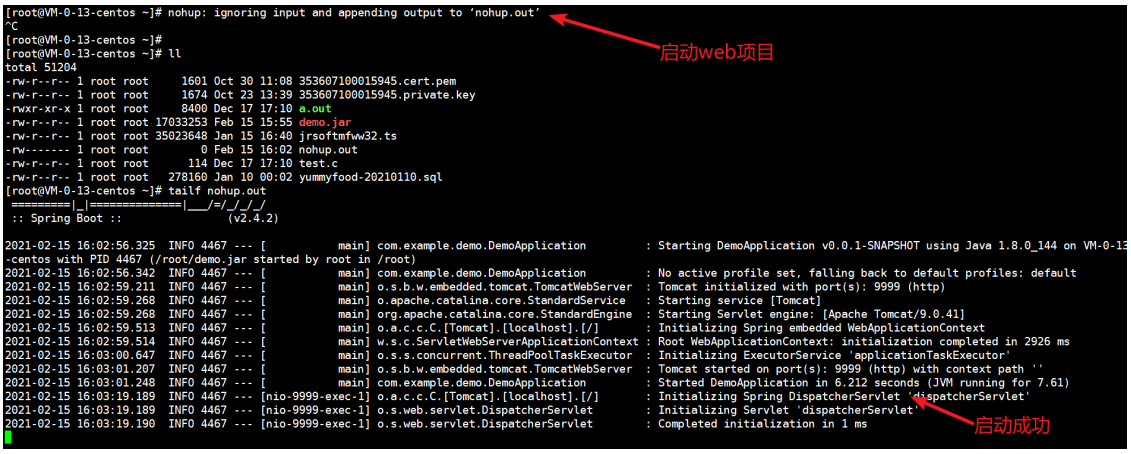
三、获取客户端真实IP

创建一个 SpringBoot 项目，写一个接口：

```
@RestController
public class TestController {

    @RequestMapping("getIpAddr")
    public String getIpAddr(HttpServletRequest request){
        String clientIp = request.getRemoteAddr();
        String clientUrl = request.getRequestURL().toString();
        return "clientIp="+clientIp+",clientUrl="+clientUrl;
    }

}
```

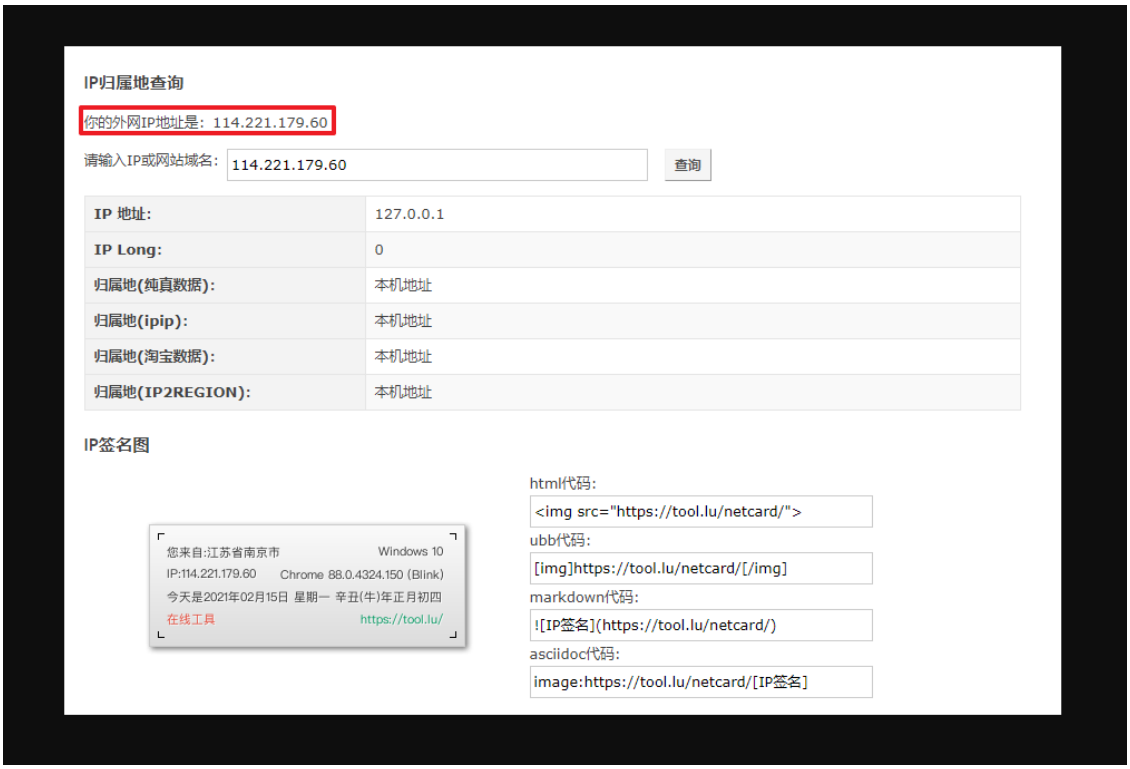
端口设置为 server.port=9999 。

下面进行 maven 打包为 xxx.jar ，上传到服务器上，服务器提前安装好 nginx 和 jdk ，直接执行 nohup java -jar xxx.jar & 即可启动服务。



我们先不通过 nginx 直接去访问下此服务，即采取客户端直接访问后端服务（注意云服务器要放开9999端口的安全组设置）

这里的 114.221.179.60 确实是我本地当前的出访IP：

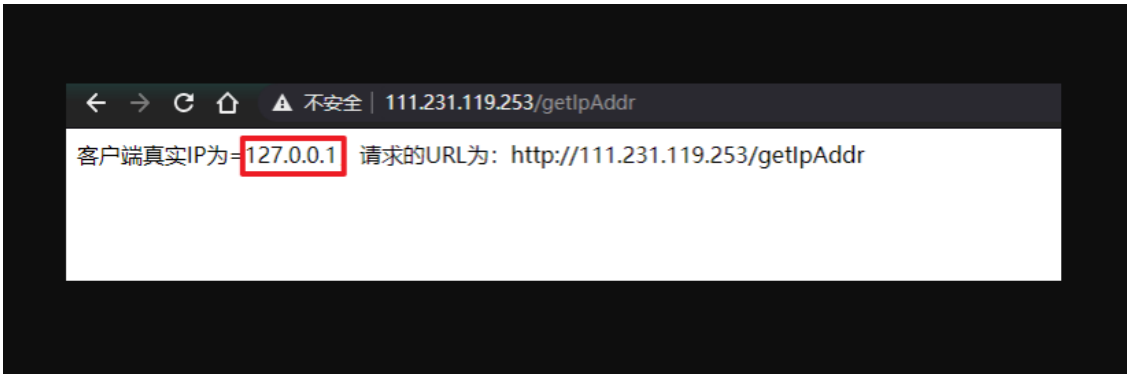


事实证明，如果没有 nginx ，我直接通过 request.getRemoteAddr() 是可以拿到客户端的真实IP地址的。

下面通过 nginx 进行访问，在 nginx 上配置代理地址，例如在我的IP为 111.231.119.253 的云服务器上， tomcat 端口号为 9999 ， Nginx 端口号 80 ， Nginx 反向代理 9999 端口：

```
server {
    listen 80;
    location / {
        proxy_pass http://127.0.0.1:9999; # 反向代理应用服务器HTTP地址
    }
}
```

在另一台机器上浏览器打开 http://111.231.119.253/getIpAddr 访问此应用，**请求就会经过nginx这个反向代理服务器，即客户端--》nginx反向代理--》真实应用服务**，获取客户端IP和URL，结果为：



佐证了上面说的：程序获取到的客户端IP是 Nginx 的IP而非浏览器所在机器IP，获取到的URL是 Nginx proxy_pass 配置的URL组成的地址，而非浏览器地址上的真实地址。

对于Web应用来说，这次HTTP请求的客户端是 Nginx 而非真实的客户端浏览器，如果不加特殊处理的话，**Web应用会把 Nginx 当做请求的客户端**，获取到的客户端信息就是 Nginx 的信息。

那么如何解决呢？办法总比困难多，需要对 nginx 做下配置才行。

```
server {
        listen       80;
        #server_name  www.oursnail.cn;

        location / {
            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_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://127.0.0.1:9999;
        }

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
```

- Host：包含客户端真实的域名和端口号。
- X-Forwarded-Proto：表示客户端真实的协议（http还是https）。
- X-Real-IP：表示客户端真实的IP
- X-Forwarded-For：这个 Header 和 X-Real-IP 类似，但是它在多层代理时会包含真实客户端及中间每个代理服务器的IP。

> $remote_addr 只能获取到与服务器本身直连的上层请求ip，所以设置$remote_addr一般都是设置第一个代理上面;但是问题是，有时候是通过cdn访问过来的，那么后面web服务器获取到的，永远都是cdn 的ip 而非真是用户ip,那么这个时候就要用到X-Forwarded-For 了。这个参数，从上面学习到，是从客户真实IP为起点，穿过多层proxy到达最终的web服务器的，所有的IP都会被记录下来，所以下面获取IP的逻辑是优先从X-Forwarded-For的IP列表中获取，获取不到才去从X-Real-IP中获取。

通过 ./nginx -s reload 进行重启，并且我需要修改代码。写一个获取IP的工具类 IpUtil ：

```
public class IpUtil {
    public static String getIp(HttpServletRequest request) throws
Exception {
        String ip = request.getHeader("X-Forwarded-For");
        if (ip != null){
            if (!ip.isEmpty() && !"unKnown".equalsIgnoreCase(ip)) {
                int index = ip.indexOf(",");
                if (index != -1) {
                    return ip.substring(0, index);//获取x-forwarded-
for中的第一个
                } else {
                    return ip;
                }
            }
        }
        ip = request.getHeader("X-Real-IP");
        if (ip != null) {
            if (!ip.isEmpty() && !"unKnown".equalsIgnoreCase(ip)) {
                return ip;
            }
        }
        ip = request.getHeader("Proxy-Client-IP");
        if (ip != null) {
            if (!ip.isEmpty() && !"unKnown".equalsIgnoreCase(ip)) {
                return ip;
            }
        }
        ip = request.getHeader("WL-Proxy-Client-IP");
        if (ip != null) {
            if (!ip.isEmpty() && !"unKnown".equalsIgnoreCase(ip)) {
                return ip;
            }
        }
        ip =  request.getRemoteAddr();
        return ip.equals("0:0:0:0:0:0:0:1") ? "127.0.0.1" : ip;
    }
}
```

新增一个接口：

```
@RequestMapping("getIpAddrByNginx")
public String getIpAddrByNginx(HttpServletRequest request){
    //获取nginx带过来的x-forwarded-for字段
    String xForwardedFor = request.getHeader("x-forwarded-for");
    String xRealIp = request.getHeader("x-real-ip");
    String clientIp = null;
    try {
        clientIp = IpUtil.getIp(request);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "x-forwarded-for的值为："+xForwardedFor+"，x-real-ip的值为："+xRealIp+"，客户端真实IP为="+clientIp;
}
```

重新发布此服务，浏览器访问：`http://111.231.119.253/getIpAddrByNginx` 结果为：

可以看到，结合上面关于 X-Forwarded-For 的介绍，我们可以知道，对于部署了 Nginx 这样反向代理的 Web 应用，在正确配置了 Set Header 行为后，可以使用 Nginx 传过来的 X-Real-IP 或 X-Forwarded-For 的第一个IP作为客户端实际的IP。

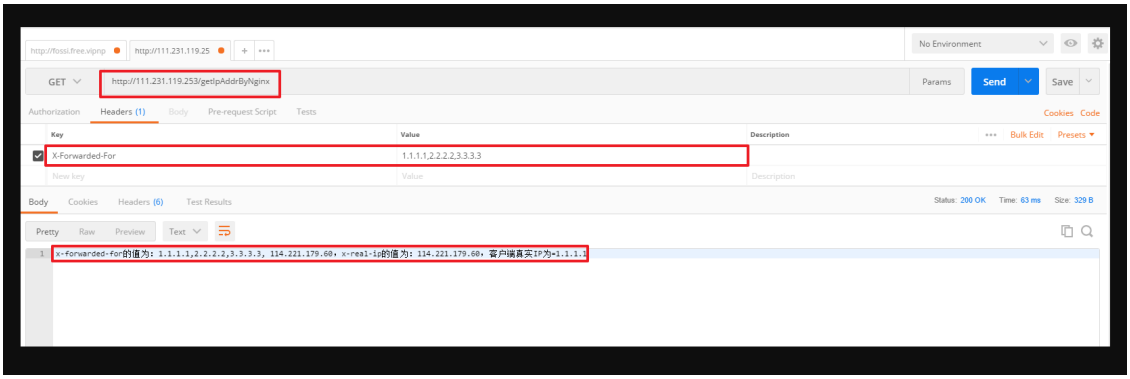
四、伪造X-Forwarded-For参数

一般客户端（例如浏览器）发送HTTP请求是没有 X-Forwarded-For 头，当请求到达第一个代理服务器时，代理服务器会加上 X-Forwarded-For 请求头，并将值设为客户端的IP地址（也就是最左边的第一个值），后面如果还有多个代理，会依次将IP追加到 X-Forwarded-For 头最右边，最终请求到达Web服务器，应用通过获取 X-Forwarded-For 头取左边第一个IP即为客户端真实IP。

正如上面所说，X-Forwarded-For 只是追加地址，就会给伪造IP可乘之机。

但是如果客户端在发起请求时，请求头上带上一个伪造的 X-Forwarded-For ，由于后续每层代理只会追加不会覆盖，那么最终到达服务器时，获取到的左边第一个IP地址将会是客户端伪造的IP。也就是上面Java代码中 getClientIp() 方法获取到的IP地址很可能是伪造的IP地址。

伪造 X-Forwarded-For 头的方法很简单，例如 Postman 就可以轻松做到：



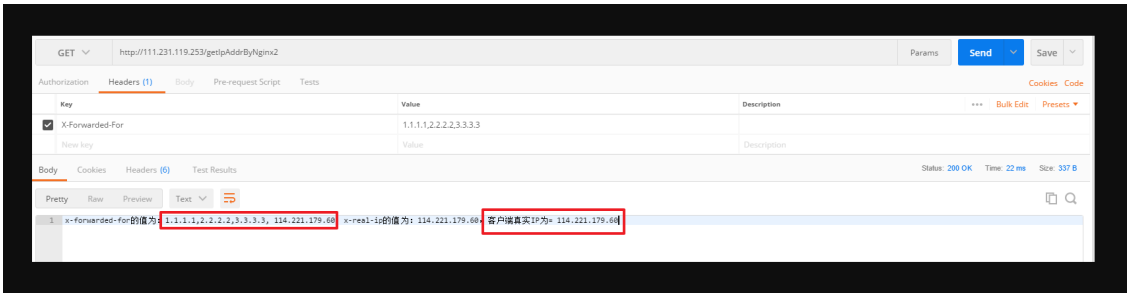
之前的代码逻辑获取到的IP就不是真实的客户端IP了，而是构造出来的第一个IP。如何解决这样的问题呢?

一个思路是：从右向左遍历。遍历时可以根据正则表达式剔除掉内网IP和已知的代理服务器本身的IP（例如192.168开头的），那么拿到的第一个非剔除IP就会是一个可信任的客户端IP。这种方法的巧妙之处在于，即时伪造X-Forwarded-For，那么请求到达应用服务器时，伪造的IP也会在X-Forwarded-For值的左边，从右向左遍历就可以避免取到这些伪造的IP地址。

比如可以通过这个工具类 IpCheckutil 来对IP做校验，结合上面的那个IPUtil使用：

```
public class IpCheckutil {
    public static final String _255 = "(?:25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)";
    public static final Pattern pattern = Pattern.compile("^(?:" +
_255 + "\\.){3}" + _255 + "$");

    public static String longToIpV4(long longIp) {
        int octet3 = (int) ((longIp >> 24) % 256);
        int octet2 = (int) ((longIp >> 16) % 256);
        int octet1 = (int) ((longIp >> 8) % 256);
        int octet0 = (int) ((longIp) % 256);
        return octet3 + "." + octet2 + "." + octet1 + "." + octet0;
    }

    public static long ipV4ToLong(String ip) {
        String[] octets = ip.split("\\.");
        return (Long.parseLong(octets[0]) << 24) +
(Integer.parseInt(octets[1]) << 16)
                + (Integer.parseInt(octets[2]) << 8) +
Integer.parseInt(octets[3]);
    }

    public static boolean isIPv4Private(String ip) {
```

```
        long longIp = ipV4ToLong(ip);
        return (longIp >= ipV4ToLong("10.0.0.0") && longIp <=
ipV4ToLong("10.255.255.255"))
                || (longIp >= ipV4ToLong("172.16.0.0") && longIp <=
ipV4ToLong("172.31.255.255"))
                || longIp >= ipV4ToLong("192.168.0.0") && longIp <=
ipV4ToLong("192.168.255.255");
    }

    public static boolean isIPv4Valid(String ip) {
        return pattern.matcher(ip).matches();
    }

    public static String getIpFromRequest(HttpServletRequest
request) {
        String ip;
        boolean found = false;
        if ((ip = request.getHeader("x-forwarded-for")) != null) {
            String[] iparr = ip.split(",");
            int len = iparr.length;
            for(int i = len-1 ; i>0 ; i--){
                //如果都是外网来访问的话，则可以从右向左遍历，排除掉内网的IP
地址，第一个非内网IP就是我们要的客户端IP，而前面伪造的IP不会被遍历到
                if (isIPv4Valid(iparr[i].trim()) &&
!isIPv4Private(iparr[i].trim())) {
                    ip = iparr[i];
                    found = true;
                    break;
                }
            }
        }
        if (!found) {
            ip = request.getRemoteAddr();
        }
        return ip;
    }
}
```



五、代理总结

从上面的测试过程中，可以看到，X-Real-IP不是一直都可以正常获取到客户端真实IP吗？你为何不依不饶地使用X-Forwarded-For，是为了用而用吗？

首先，如果有多级代理，x-forwarded-for效果是大于x-real-ip的，可以记录完整的代理链路。

更重要的是，X-Forwarded-For是在存在正向代理、反向代理情况下的标准用法，而正向代理中是没有x-real-ip相关的标准的。**

前面介绍了三种获取的方式：request.getRemoteAddr()、X-Real-IP以及X-Forwarded-For三者中如何选择呢？

It denpends on how much you know the network between the client and the server, and how much you trust these headers.

当你确定客户端和服务端之间是直连的，无任何代理，那么就直接使用request.getRemoteAddr()。

当架构如我们的文章一样，是客户端---》nginx反向代理---》真实应用，无其他任何组件，并且nginx的配置也是如此配置的话，那么就可以使用X-Real-IP。

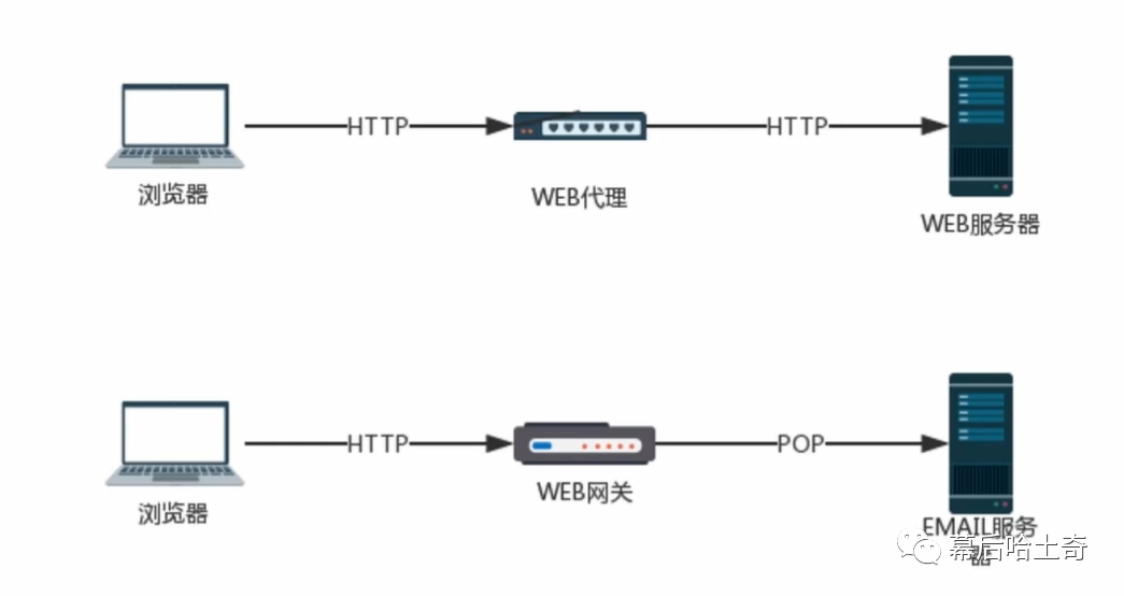
但是当你不清楚客户端和真实应用之间到底经历了多少代理时，请使用X-Forwarded-For获取。

但是，一切都会存在意外，具体的代码获取IP还得根据实际代理情况去调整，因此不是说代码直接copy就可以拿来用的，这也是前文中为什么说获取IP需要格外的小心！

六、HTTP中介之网关

- 网关可以作为某种翻译器使用，它抽象出了一种能够到达资源的方法。网关是资源和应用程序之间的粘合剂。
- 网关扮演的是“协议转换器”的角色。

网关和代理最大的区别是：网关可以进行协议转换。



如上图，客户端发送HTTP请求，咱们的网关将其转换为邮件协议。

有时用于HTTP和HTTPS的转换，有的时候将统一对外提供资源的一个流量入口统称为网关，这个入口可以进行身份认证、权限认证等，然后进行流量转发，不用太过纠结其概念区别，重要的是灵活使用嘛。