HTTP代理

在此作业中，您将实现一个 Web 代理，用于在多个 Web 客户端和 Web 服务器之间传递请求和数据。代理应该支持**并发**连接。这项作业将使您有机会了解 Internet 上最流行的应用程序协议之一——超文本传输协议 (HTTP)。完成分配后，您应该能够配置 Firefox 以使用您的代理实现。

简介：超文本传输协议(The Hypertext Transfer Protocol)

超文本传输协议 (HTTP) 是用于 Web 通信的协议：它定义 Web 浏览器如何从 Web 服务器请求资源以及服务器如何响应。为简单起见，在本次作业中，我们将仅处理 HTTP 协议 1.1 版本，该协议在 RFC 2616中有详细定义。您无需阅读全文即可完成此作业。我们进一步总结了最重要的部分。

HTTP 通信以事务的形式发生；事务包括客户端向服务器发送请求，然后读取响应。请求和响应消息共享通用的基本格式：

- 初始行（ initial line，请求或响应行，定义如下）
- 零个或多个标题行（header line）
- 空行 (blank line， CRLF)
- 可选的消息正文。

初始行和标题行后面各跟着一个“回车换行符”(\r\n)，表示行尾。

对于大多数常见的 HTTP 事务，该协议可归结为一系列相对简单的步骤（RFC 2616的重要部分位于括号中）：

1. 客户端创建到服务器的连接。
2. 客户端通过向服务器发送一行文本发出请求。这**请求行**包括一个HTTP *方法*（通常是 "GET"，但也可能是 "POST"、"PUT" 等），一个 *请求 URI*（类似于 URL）和客户端希望使用的协议版本（"HTTP/1.1"）。请求行后面跟着一个或多个头部行。初始请求的消息体通常为空。（5.1-5.2，8.1-8.3，10，D.1）
3. 服务器发送一个响应消息，其初始行包含一个**状态行**，指示请求是否成功。状态行包括HTTP版本（"HTTP/1.1"）、*响应状态码*（一个数字值，指示请求是否成功完成）和一个*原因短语*，是一个提供状态码描述的英语消息。与请求消息一样，响应中的头字段可以有很多或很少，具体取决于服务器要返回的内容。在CRLF字段分隔符之后，消息体包含了在请求成功时客户端请求的数据。（6.1-6.2，9.1-9.5，10）
4. 一旦服务器将响应返回给客户端，它就会关闭连接。

无需使用网络浏览器即可轻松查看此过程的运行情况。从您的终端输入：

(无法运行请检查电脑的telnet服务是否启动)

```
telnet www.baidu.com 80
```

这将打开一个到www.baidu.com服务器的 TCP 连接，该连接监听端口 80（默认 HTTP 端口）。回车后您会进入一个空白界面，输入以下命令：

```
GET / HTTP/1.1
```

并按回车键**两次**。您应该看到类似以下内容：

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
```

```
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 9508
Content-Type: text/html
Date: Wed, 22 Nov 2023 14:57:19 GMT
P3p: CP=" OTI DSP COR IVA OUR IND COM "
P3p: CP=" OTI DSP COR IVA OUR IND COM "
Pragma: no-cache
Server: BWS/1.1
Set-Cookie: BAIDUID=F95FEC4A729258E6B24B2D5CFF0547D8:FG=1; expires=Thu, 31-Dec-
37 23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: BIDUPSID=F95FEC4A729258E6B24B2D5CFF0547D8; expires=Thu, 31-Dec-37
23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: PSTM=1700665039; expires=Thu, 31-Dec-37 23:55:55 GMT; max-
age=2147483647; path=/; domain=.baidu.com
Set-Cookie: BAIDUID=F95FEC4A729258E673EFB3B851AE40B4:FG=1; max-age=31536000;
expires=Thu, 21-Nov-24 14:57:19 GMT; domain=.baidu.com; path=/; version=1;
comment=bd
Traceid: 1700665039240463975411257475731430361673
Vary: Accept-Encoding
X-Ua-Compatible: IE=Edge,chrome=1

<!doctype html><html itemscope="" ...  (More HTML follows)
```

还可能有一些额外的标头信息，例如设置 cookie 和/或对浏览器或代理的缓存行为的指令。您所看到的正是您的网络浏览器在访问 Baidu 主页时看到的内容：HTTP 状态行、标头字段，以及最后由浏览器解释以创建网页的 HTML 组成的 HTTP 消息正文。

HTTP 代理（Proxy）

通常，HTTP 是一种客户端-服务器协议。客户端（通常是您的网络浏览器）直接与服务器（网络服务器软件）通信。然而，在某些情况下，引入称为代理的中间实体可能很有用。从概念上讲，代理位于客户端和服务器之间。在最简单的情况下，客户端不是将请求直接发送到服务器，而是将其所有请求发送到代理。然后代理打开与服务器的连接，并传递客户端的请求。代理接收来自服务器的回复，然后将该回复发送回客户端。请注意，代理本质上就像 HTTP 客户端（对于远程服务器）和 HTTP 服务器（对于初始客户端）一样。

为什么要使用代理？有以下几个可能的原因：

- **性能**：通过保存其获取的页面的副本，代理可以减少创建与远程服务器的连接的需要。这可以减少检索页面所涉及的总体延迟，特别是在服务器位于远程或负载较重的情况下。
- **内容过滤和转换**：虽然在最简单的情况下，代理仅获取资源而不检查它，但没有任何内容表明代理仅限于盲目获取和提供文件。代理可以检查请求的 URL 并有选择地阻止对某些域的访问、重新格式化网页（例如，通过删除图像以使页面更容易在手持设备或其他资源有限的客户端上显示），或执行其他转换和过滤。
- **隐私**：通常，Web 服务器会记录所有传入的资源请求。此信息通常至少包括客户端的 IP 地址、浏览器或他们正在使用的其他客户端程序（称为用户代理）、日期和时间以及请求的文件。如果客户端不希望记录此个人身份信息，通过代理路由 HTTP 请求是一种解决方案。来自使用同一代理的客户端的所有请求似乎都来自代理本身的 IP 地址和用户代理，而不是各个客户端。如果多个客户端使用相同的代理（例如，整个企业或大学），则将特定 HTTP 事务链接到单个计算机或个人会变得更加困难。

A 部分：HTTP 代理

入门

对于 VirtualBox 用户

在您的主机（不是虚拟机）上，转到课程作业目录：

```
$ cd COS461-Public/assignments
```

从 Github 拉取更新。

```
$ git pull
```

无需重新配置您的虚拟机。`vagrant ssh`如果已经配置了虚拟机，您可以直接进入该虚拟机。但是，如果您在 git pull 时看到 Vagrantfile 已更新，则需要重新配置虚拟机以安装此分配的软件包，如下所示：

```
$ vagrant reload --provision
```

然后，通过 ssh 进入虚拟机。

```
$ vagrant ssh
```

对于 Apple Silicon 用户

`ssh`进入您的 UTM VM，从 Github 获取最新更新：

```
$ git pull
```

如果您发现该`setup.sh`文件已更新，您可以手动应用差异或删除当前的虚拟机（确保您的进度已备份在主机上），并按照分配 1 自述文件中的设置过程来反映更改。如果未更新，则无需对您的虚拟机执行任何操作。

任务规范

您的任务是构建一个能够接受 HTTP 请求、将请求转发到远程（原始）服务器以及将响应数据返回到客户端的 Web 代理。

代理将在 Go 文件中实现`http_proxy.go`。 **允许并鼓励您使用 Go`net`和`net/http`库**，文档位于：https://golang.org/pkg/net/ &https://golang.org/pkg/net/http/。理解并正确使用这些库将使这项任务变得更加简单。

HTTP是一种在TCP之上运行的应用层协议。这使您能够在多个抽象级别进行编程，因为所有HTTP都只是TCP数据包的一部分。`net`库包含在传输层（TCP）操作的套接字编程函数和类型。`net`库的函数更通用，总体需要更多的代码和手动创建HTTP消息字符串，但它们更灵活且文档更清晰。`http`库包含在HTTP应用层操作的函数和类型。`http`库的函数更专注于HTTP，可能使您使用更少的代码行，但它们较不灵活（并且文档更难理解）。

我们的参考解决方案使用`http`库中的`Request`类型和相关函数来解析和修改HTTP请求，但使用`net`中的`Listen()`、`Accept()`和`Dial()`函数（而不是`http`中的各种服务器和客户端函数）。这个参考实现大约有130行代码。最终决定如何结合`net`和`http`库来实现代理取决于你。只要不影响代理行为的实现选择不会影响你的评分。

你的代理应该在课程虚拟机上使用提供的`Makefile`编译时没有错误。编译应该生成一个名为`http_proxy`的二进制文件，它的第一个参数应该是要监听的端口。不要使用硬编码的端口号。

您不应假设您的代理将在特定 IP 地址上运行，或者客户端将来自预先确定的 IP 地址。

获取客户的请求

你的代理应该在指定的端口上进行监听并等待传入的客户端连接。请查看assignment 1中的Go socket文档以获取有关在Go中进行套接字编程的更多信息。客户端请求应该被并发处理，每个请求都应该生成一个新的gouroutine来处理。

一旦客户端连接，代理应该从客户端读取数据，然后检查是否有格式正确的HTTP请求。使用`http`库确保代理收到包含有效请求行的请求。

你的代理只负责处理GET方法。所有其他请求方法应该引发一个带有状态码500 "Internal Error"的格式良好的HTTP响应。这篇维基百科文章有一个完整的有效HTTP状态码列表：https://en.wikipedia.org/wiki/List_of_HTTP_status_codes。你可能不需要返回状态码418 "I'm a teapot"...

向服务器发送请求

一旦代理解析了客户端请求中的URL，它可以与请求的主机建立连接（使用适当的远程端口，如果未指定，则使用默认的80端口），并发送用于请求适当资源的HTTP请求。代理应该始终以相对URL + Host头的格式发送请求，而不管请求是如何从客户端接收的。

例如，如果代理接受来自客户端的以下请求：

```
GET http://www.princeton.edu/ HTTP/1.1
```

它应该向远程服务器发送以下请求：

```
GET / HTTP/1.1
Host: www.princeton.edu
Connection: close
(Additional client specified headers, if any...)
```

请注意，我们总是向服务器发送HTTP/1.1标志和一个`Connection: close`头，以便在其响应完全传输后关闭连接，而不是保持持久连接。因此，虽然你应该将从客户端接收到的客户端头传递给服务器，但你应该确保替换从客户端接收到的任何`Connection`头，将其替换为指定`close`，如上所示。要添加新的头部或修改现有的头部，使用`http`库的`Request`类型。

向客户返回响应

在从远程服务器接收到响应后，代理应该通过适当的套接字将响应消息原样发送给客户端。为了严格遵循规范，代理应确保服务器的响应中存在`Connection: close`，以便让客户端决定是否在接收响应后关闭其端口。然而，在这个作业中，不要求检查这一点，因为一个表现良好的服务器会在我们确保代理向服务器发送一个关闭token时回应一个`Connection: close`。

从代理到客户端的状态码

对于代理捕获的任何错误，代理应返回状态 500 'Internal Error' 。如上所述，除 GET 之外的任何请求方法都应导致您的代理返回状态 500 'Internal Error' 而不是 501 'Not Implemented' 。同样，对于任何无效、格式不正确的标头或请求，您的代理应向客户端返回状态 500 'Internal Error' 而不是 400 'Bad Request' 。

否则，您的代理应该简单地将状态回复从远程服务器转发到客户端。这意味着大多数 1xx、2xx、3xx、4xx 和 5xx 状态回复应通过代理直接从远程服务器发送到客户端。（在调试时，请确保来自远程服务器的 404 状态回复不是来自代理的请求转发不当的结果。）

测试你的代理

构建您的可执行文件：

```
go build http_proxy.go
```

使用以下命令运行代理：

`./http_proxy &`，其中`port`是代理应侦听的端口号。作为功能的基本测试，尝试使用 telnet 请求页面（不要忘记按两次 Enter 键）：

```
telnet localhost <port>
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
GET http://www.example.com/ HTTP/1.1
```

如果你的代理工作正常，example.com的头部和HTML应该显示在你的终端屏幕上。请注意，在这里，我们请求的是绝对URL（`http://www.example.com/`），而不仅仅是相对URL（`/`）。检查代理行为的一个很好的检查方法是比较通过你的代理获取的HTTP响应（头部和正文）与通过直接telnet连接到远程服务器获取的响应。另外，尝试从两个不同的shell并发地使用telnet请求一个页面。

然后，尝试使用提供的`test_proxy.py`脚本测试你的代理。这将比较直接获取3个预定网站的结果与通过你的代理获取的结果：

```
python test_scripts/test_proxy.py http_proxy  <port (optional, will be random if
omitted)>
```

在通过`test_proxy.py`中的所有测试之后，尝试运行`test_proxy_conc.py`脚本。这将使用不同数量的并发客户端连接测试你的代理。

```
python test_scripts/test_proxy_conc.py http_proxy  <port (optional, will be
random if omitted)>
```

对于稍微复杂的测试，您可以将 Firefox 配置为使用您的代理服务器作为其 Web 代理，如下所示：

1. 在端口 12000 上运行代理。
2. 在您的主机上，打开 Firefox
3. 转到“首选项”。选择“高级”，然后选择“网络”。
4. 在“连接”下，选择“设置...”。
5. 选择“手动代理配置”。删除默认的“无代理：localhost 127.0.0.1”。
6. 在“HTTP 代理”下输入代理程序运行的主机名 (127.0.0.1) 和端口 (12000)。
7. 通过在连接选项卡中选择“确定”来保存更改，然后在首选项选项卡中选择“关闭”。

B 部分：通过 DNS 预取进行代理优化

DNS 预取是一种技术，在用户单击任何链接之前，Web 代理可以解析 HTTP 响应中嵌入的链接的域名。通过将这些链接的 DNS 条目放入最靠近代理的 DNS 解析器的缓存中，可以缩短页面加载时间。DNS 预取依赖于一个简单的预测：如果用户请求特定页面，他或她接下来很可能会请求从该页面链接到的页面。

任务规范

你的任务是在你在Part A中实现的代理中添加DNS预取。首先，复制你的代理，将其命名为`http_proxy_DNS.go`：

```
cp http_proxy.go http_proxy_DNS.go
```

你应该在这个新文件中实现DNS预取。**不要编辑原始代理**，因为你需要提交两个版本。在新复制的文件中，更改头部中的文件名为"`http_proxy_DNS.go`"。

为了找到远程服务器响应中的链接，你需要解析HTML内容。`net/html`库包含用于标记化HTML的函数：https://godoc.org/golang.org/x/net/html。找到HTML中所有以"http"开头的`<a>`标签的`href`属性。然后，使用`net`库中的`LookupHost()`函数为每个链接发出DNS查询。解析HTML和发送DNS查询应该在新的goroutines中进行，以不减慢向客户端返回响应的速度。

请注意，你无需处理DNS响应。仅仅发送它们将填充DNS解析器的缓存。

与之前一样，你的DNS预取代理应该在课程虚拟机上使用提供的`Makefile`编译时没有错误。编译应该生成一个名为`http_proxy_DNS`的二进制文件，它的第一个参数应该是要监听的端口。不要使用硬编码的端口号。你不应该假设你的代理将运行在特定的IP地址上，或者客户端将来自预定的IP地址。

测试您的 DNS 预取代理

构建您的可执行文件：

```
go build http_proxy_DNS.go
```

按照与测试原始代理相同的方式测试您的 DNS 预取代理。使用测试脚本时，只需更改第一个命令行参数：

```
python test_scripts/test_proxy.py http_proxy_DNS  <port (optional, will be
random if omitted)>
python test_scripts/test_proxy_conc.py http_proxy_DNS <port (optional, will be
random if omitted)>
```

在这些测试中，你不会注意到任何加速，因为它们不通过链接访问其他网站。

此外，你可以使用Wireshark来验证代理是否正确执行DNS查找。

最后，尝试像以前一样使用你的新`http_proxy_DNS`与Firefox。希望你会注意到通过当前页面上的链接访问的页面加载更快。然而，你看到的加速程度取决于你本地DNS解析器和网络连接的速度，所以如果没有明显感觉到，也不要感到惊讶。