深入 HTTP 协议以及 Web 运维

—— Web 端运维基础知识

Victor Huang | Mar. 26th, 2022

目录

- HTTP 协议和 Web 浏览器
- Web 运维部分

HTTP 协议和 Web 浏览器

HTTP 的演进

- HTTP/0.9：单行协议：
  - 请求只能容纳一行，只有 GET 方法。例如 `GET /index.html`。
  - 响应体只包含文件内容。
- 不足：
  - 没有 HTTP 头机制，无法传递除文件之外的任何信息。
  - 没有状态码机制，不易判断是否出现错误。
  - 无法进行文件上传。

HTTP 的演进

- HTTP/1.0：
  - 加入了 HTTP 头机制，拥有了可扩展性。
  - 引入了 HTTP 状态码，易于查错。
  - 引入了 POST 等方法。
- 不足：
  - 标准制定时歧义较大，实现较为混乱。
  - 一个连接只能进行一次请求和响应。

HTTP 的演进

- HTTP/1.1：
  - 加入了连接复用机制，可以在一个 TCP 连接中收发多个 HTTP 请求和响应。
  - 加入了 HTTP Pipelining，客户端可以连续发送请求。
  - 响应分块、编码控制等等一系列改进。
- 不足：
  - 纯文本协议传输和解析开销相对都偏高，且 HTTP 头无法压缩。
  - 请求和响应必须顺序抵达，等待开销较大。

HTTP 的演进

- HTTP/2：
  - 全面采用二进制协议，降低了机器传输和解析开销。
  - HTTP 头可以被压缩。
  - 引入了 Server Push，服务器可以主动向客户端推送数据。
  - 多种优化措施，例如连接的合并。
- 不足：
  - 仍然基于 TCP 连接，受到三次握手的限制，延时较高，且存在队列阻塞的问题。

HTTP 的演进

- HTTP/3：
  - 改用 UDP 传输，避免了 TCP 握手延时和队列阻塞。
  - 采用 QUIC 连接控制，配合 TLS 1.3 可以实现 0-RTT 连接。
- 不足：
  - UDP 在部分地区易受限制。
  - 客户端、网络设备支持情况目前较差，未来可期。

HTTP 的基本性质

- 无状态，有会话
  - HTTP 本身是无状态协议，会话通过 Cookie 保持
- 简单而可扩展
  - HTTP/2 之前的 HTTP 为纯文本协议
  - 通过 HTTP 头的机制，实现各种扩展功能

HTTP 重定向

- 301、302、303、307、308 是重定向状态码。

	缓存（永久重定向）	不缓存（临时重定向）
转GET	301	302、303
方法保持	308	307

- 使用 Location 响应头来返回目标地址。
  - 一种返回格式是完整 URL，例如 `Location: https://redrock.team/`
  - 另一种是返回路径，例如 `Location: /foo/`

HTTP 连接复用

- HTTP/1.1 加入了长连接。
  - Connection 头配置为 keep-alive 时，HTTP 进入长连接状态。服务端发送完响应后不关闭连接。客户端可以在同一个连接内发送请求。
  - 结束连接或明确表示不启用长连接，Connection 头配置为 close。
- HTTP/2 在多个连接的域名可被同一 HTTPS 证书验证时，会自动合并连接来减少连接数，但这在某些情况下可能导致问题。
  - https://www.zhihu.com/question/310263956/answer/601458852

HTTP 会话

- HTTP 通过 Cookie 保持会话。
- Cookie 是一种 Key-Value 集合，并附加了一些配置项目。接受 Cookie 的客户端每次请求时带着 Cookie 头发给服务器。
- 服务端可以通过 Set-Cookie 头来设置 Cookie。
  - 例如 `Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly`
- Cookie 有许多属性，例如上面的：
  - Expires：设置了 Cookie 的过期时间
  - Secure：只允许在 HTTPS 下传输
  - HttpOnly：不允许通过 JavaScript 脚本读取 Cookie。

HTTP 会话

- Cookie 可以通过属性限制它被发往哪里：
  - Domain 属性指定了哪些域名可以被发送 Cookie，它包括子域名。即 Domain=redrock.team 允许向 app.redrock.team 发送 Cookie。
  - Path 属性指定了哪些路径可以被发送 Cookie。
  - SameSite 限制跨站发送 Cookie。

HTTP 内容传输

- 众所周知，一个网站有大量的资源，对于不同的资源，浏览器的处理方式也应当不同。
- HTTP 头 Content-Type 则表示了资源的类型。例如：
  - HTML 网页是 text/html
  - PNG 图片是 image/png
- 这种文件类型我们称之为 MIME 类型。它的结构就是：
  - 类型/具体类型
- 具体的 MIME 类型介绍：https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/MIME_types

HTTP 内容传输

- Content-Type 头既可以出现在请求中又可以出现在响应中。
  - 对于请求，它表示 POST 数据时请求体的数据格式。
    - 对于表单，通常有两种格式，一种是 application/x-www-form-urlencoded，另一种是 multipart/form-data。
      - 第一种:

```
POST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

say=Hi&to=Mom
```

      - 第二种:

```
POST /test.html HTTP/1.1
Host: example.org
Content-Type: multipart/form-data;boundary="boundary"

--boundary
Content-Disposition: form-data; name="field1"

value1
--boundary
Content-Disposition: form-data; name="field2"; filename="example.txt"

value2
```

HTTP 内容传输

- 对于响应，它表示响应体的格式。
- 有时，在 MIME 类型后面，可能会跟着它的编码。

- 我们在下载时可以看到文件的大小，这是由 Content-Length 头控制的。Content-Length 头的值是一个数字，表示传输文件的字节数。
  - Content-Length 头也会在携带 Body 的请求体出现。
- 偶尔，我们希望改变内容的呈现方式，这时 Content-Disposition 头则处理这类问题。
  - 作为响应时，Content-Disposition: attachment 表示强制作为下载，inline 则期望浏览器尽量不用下载弹框
  - Content-Disposition 还有 filename 属性，可以指定文件名。

```
Content-Disposition: inline; filename=equation-' . $random . '.emf'
```

HTTP 内容传输

- HTTP 有一系列 Accept 的头供客户端传给服务端，以声明其想要的是怎样的数据。
  - Accept-Encoding 头控制期望的数据编码，因此它可以请求是否使用压缩。例如使用 `Accept-Encoding: gzip, deflate, br` 则告诉服务端可以使用 gzip、deflate、br 等压缩算法。
  - Accept 头说明期望的数据格式，例如 Accept: image/png 则期望服务端发送 PNG 图片。

HTTP 内容传输

- HTTP 具有断点续传的能力。
  - Range 头可以供客户端指定请求的数据范围，格式是

    ```
    Range: (单位=起始)-[终止(闭区间)]
    ```

  - 服务器收到后，断点续传的内容会返回状态码 206 Partial Content，并携带 Content-Range 头表示当前传输的范围。

    ```
    Content-Range: 单位 起始-终止(闭区间)/总长度
    ```

  - 如果终止范围超出文件长度，则认为请求不合法，返回 416 Requested Range Not Satisfiable。
  - 如果终止范围小于起始范围，传输整个文件返回 200。
  - 例如 Range=-114 这一类请求头，表示获取文件最后的 114 个字节。

HTTP 内容传输

- HTTP 还具有条件请求的能力。为了减少不必要的数据传输，客户端可以携带对本地缓存资源的描述，若资源没有更改则无需传输。
  - 服务端返回资源时，可以在头部附带 Last-Modified 头表示最后一次修改的时间，或者 ETag 来表示文件的“实体值”，作为资源当前状态的唯一描述。
  - 客户端请求时，可以带上 If-Modified-Since 头或 If-Not-Match 头，分别对应 Last-Modified 和 ETag。
  - 如果服务端的资源发生变更，则 200 返回新的资源并更新上述头。否则，返回 304 Not Modified 告诉客户端无需更改并不传输资源。
- HTTP 也支持控制客户端缓存。
  - Cache-Control 头决定了缓存如何进行控制。
  - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Cache-Control

HTTP Referer

- 在浏览器中，用户从链接打开网页以及页面内引入资源时，会带上 Referer 表明来源。
- 通常，可以通过 Referer 统计用户的来源，也可以通过 Referer 限制来防盗链。
- Referer 其实是一个错误拼写，正确的拼写是 Referrer。但 Referer 这个拼写错误在被应用需求后才被发现，为了兼容现有实现，Referer 被将错就错地纳入了标准。
  - 但在 Web 浏览器的 JavaScript 中，访问 Referer 采用的是 document.referrer，一定程度上避免了该问题的进一步传播。

HTTP 验证

- 访问被保护的资源时，对于尚未授权的客户端服务器返回 401 Unauthorized，同时返回 WWW-Authenticate 头表示支持的验证方式。

  `WWW-Authenticate: 类型 额外参数`

- 客户端携带 Authorization 头，Authorization 头中包含授权信息。

  `Authorization: 类型 授权字符串`

- 目前最常见的验证方式有两种：
  - HTTP 基本验证
    - Authorization 头携带 Basic “用户名:密码"的base64编码 格式
  - Bearer 令牌验证
    - Authorization 头携带 Bearer Token 格式

同源策略和跨域资源共享

- 随着 Web 技术的发展，网页可以通过 JavaScript 进行异步请求访问各种资源。但是如果允许发送任意请求可能会存在安全问题。
- 因此浏览器进行了"同源策略"。
  - https://developer.mozilla.org/zh-CN/docs/Web/Security/Same-origin_policy
- 但现实业务中，我们不太可能只从同源加载资源。因此为了让资源可以安全地跨域加载，跨域资源共享 (CORS) 被提出。
  - https://developer.mozilla.org/zh-CN/docs/Web/HTTP/CORS

同源策略和跨域资源共享

- CORS 预检请求：
  - 使用 OPTIONS 方法。
  - 携带 Origin。
  - 携带 Access-Control-Request-Method 和 Access-Control-Request-Headers
- CORS 预检响应：
  - 返回 204 No Content。
  - Access-Control-Allow-Origin: 允许的源
  - Access-Control-Allow-Methods: 允许的方法
  - Access-Control-Allow-Headers: 允许携带的头
  - Access-Control-Max-Age: 以上信息的客户端缓存时间

WebSocket

- HTTP 归根结底还是必须由客户端发起请求，这就导致服务器难以主动向客户端推送消息。
- 因此如果遇到类似聊天室或在线游戏等状态实时变化的场景，单纯的 HTTP 便有些难过，常用方式有：
  - 轮询：不断发起请求向服务器询问是否有新的内容，代价是带来了巨大的并发压力。
  - 长连接：发起连接后服务器不返回数据，直至有新的内容。但这种方式仍然不能做到连续推送数据。
- 因此，为了解决这个问题，WebSocket 出现了。WebSocket 是一个类似 TCP 的全双工协议，双方可以自由收发数据。

WebSocket

- WebSocket 在握手阶段使用 HTTP 协议，但完成握手后就仅仅是使用 HTTP 所建立的连接来传输数据，和 HTTP 协议再无关联。
- WebSocket 连接建立的过程：
  - 客户端请求服务器，携带头部如下：
    - Connection: upgrade　　// 表示升级协议
    - Upgrade: websocket　　// 表示升级到 WebSocket
    - Sec-WebSocket-Version: 13　　// 表示使用的 WebSocket 版本
    - Sec-WebSocket-Key: 一段随机生成的密钥

WebSocket

- 服务器返回 101 Switching Protocols，并返回以下头。
  - Connection: Upgrade
  - Upgrade: websocket
  - Sec-WebSocket-Accept: 从客户端 Sec-WebSocket-Key 计算得
- 之后便建立起了全双工通信。

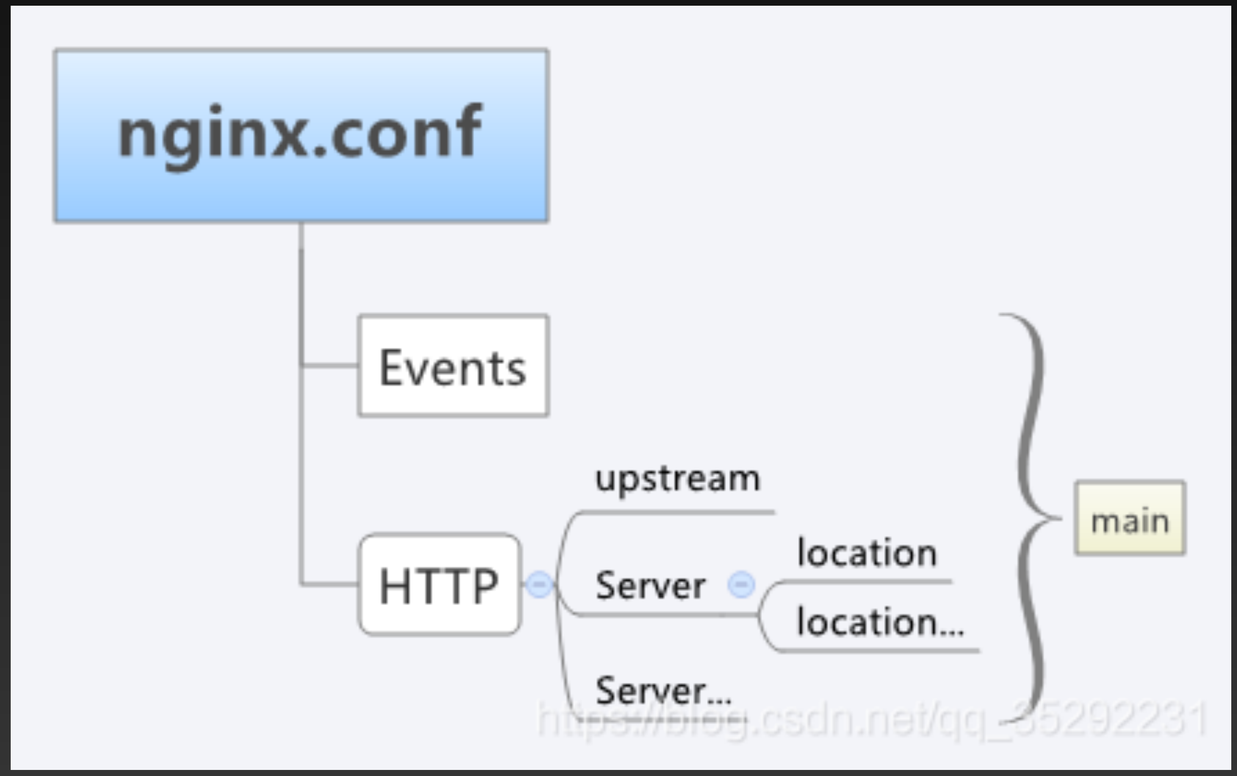
Web 运维部分

反向代理

- 反向代理：作为入口接受用户请求，并把请求转发到真正的后端。在反向代理过程中可以对请求、响应进行修改。
- 我们接下来将会以 Nginx 为例进行讲解。Nginx 是一个优秀的 web 服务器和反向代理。

Nginx

- Nginx 的配置文件以 nginx.conf 为入口。结构大概如下：



虚拟主机

- Nginx 的一个站点称为一个虚拟主机。

```
server {
    listen 80;
    server_name portainer.lyoi.cc;
    location / {
        proxy_pass http://127.0.0.1:20171;
        proxy_redirect      off;
        proxy_set_header    Host                $http_host;
        proxy_set_header    X-Forwarded-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    Upgrade             $http_upgrade;
        proxy_set_header    Connection          $connection_upgrade;
        proxy_http_version  1.1;
    }
}
```

- 以上是一个简单的反向代理的虚拟主机。
  - $ 开头的在 nginx 中称为变量。
  - nginx 有一些自带变量，你也可以自行设置变量。
  - HTTP 头可以通过 $http_头名 来获取，其中 - 替换为 _

Location 匹配

模式	含义
location = /uri	= 表示精确匹配，只有完全匹配上才能生效
location ^~ /uri	^~ 开头对URL路径进行前缀匹配，并且在正则之前。
location ~ pattern	~ 开头表示区分大小写的正则匹配
location ~* pattern	~* 开头表示不区分大小写的正则匹配
location /uri	不带任何修饰符，也表示前缀匹配，但是在正则匹配之后
location /	通用匹配，任何未匹配到其它 location 的请求都会匹配到，相当于 switch 中的 default

如何在反代下获取访客 IP

- 在 CDN 或反代后面，我们得到的访客 IP 是代理的 IP。
- 我们通常使用 X-Forwarded-For header 来获取访客 IP。
  - X-Forwarded-For 每经过一个代理便会被追加一次。
  - 它的格式：X-Forwarded-For: client, proxy1, proxy2
- 其它 header 例如 X-Real-IP 也可以用于传递 IP 信息。
- Nginx 的 Real IP 模块经过配置后，可以从 X-Forwarded-For 中递归提取 IP。
  - set_real_ip_from CIDR：信任来自这些主机的转发 IP
  - real_ip_header 指定使用哪个 header
  - real_ip_recursive on/off：指定是否递归提取 X-Forwarded-For
- 扩展：X-Forwarded-Proto 通常被定义为访客使用的协议 (http/https)

其它讲解内容清单

- Map 变量
- 多 upstream
- rewrite

ACME

- Let's Encrypt 提出了 ACME 来自动签发 SSL 证书。
- ACME 协议支持的几种验证方式：
  - HTTP-01 验证方式：
    - CA 提供一个 token 和 fingerprint，然后试图访问你的网站的以下地址，验证指纹是否匹配
    - /.well-known/acme-challenge/<TOKEN>
    - 这种方式不能签发通配符的泛域名证书。
  - TLS-ALPN-01 验证方式：
    - 目前应用不广泛，使用 TLS 握手时的 ALPN 字段验证。

ACME

- DNS-01 验证方式：
  - 通过验证 _acme-challenge.<YOUR_DOMAIN> 的 TXT 记录来证明域名的所有权。
  - 允许使用 CNAME，这被称为 DNS Alias 模式。
  - 可以签发泛域名的通配符证书。
- 常用工具：certbot、acme.sh 等

作业

- Lv0. 自行练习 nginx 使用，部署一个基于 websocket 的服务，例如 jupyterlab。
- Lv1. 写一个类似 https://cors-container.herokuapp.com 的 CORS 代理，用于解决后端没有提供跨域的情况。
- Lv2. 自己写一个 http server，加分项是支持 304 条件请求。

- 尽力去做，不设强制要求。
- 加入 GitHub Classroom: https://classroom.github.com/a/8RZ7Lj7K