一 、基础概念

URL

URI 包含 URL 和 URN，目前 WEB 只有 URL 比较流行，所以见到的基本都是 URL。

URI（Uniform Resource Identifier，统一资源标识符）

URL（Uniform Resource Locator，统一资源定位符）

URN（Uniform Resource Name，统一资源名称）

二、HTTP 方法

客户端发送的 请求报文 第一行为请求行，包含了方法字段。

GET

获取资源

当前网络请求中，绝大部分使用的是 GET 方法。

HEAD

获取报文首部

和 GET 方法一样，但是不返回报文实体主体部分。

主要用于确认 URL 的有效性以及资源更新的日期时间等。

POST

传输实体主体

POST 主要用来传输数据，而 GET 主要用来获取资源。

更多 POST 与 GET 的比较请见第九章。

PUT

上传文件

由于自身不带验证机制，任何人都可以上传文件，因此存在安全性问题，一般不使用该方法。

PATCH

对资源进行部分修改

PUT 也可以用于修改资源，但是只能完全替代原始资源，PATCH 允许部分修改。

DELETE

删除文件

与 PUT 功能相反，并且同样不带验证机制。

OPTIONS

查询支持的方法

查询指定的 URL 能够支持的方法。

会返回 Allow: GET, POST, HEAD, OPTIONS 这样的内容。

CONNECT

要求在与代理服务器通信时建立隧道

使用 SSL（Secure Sockets Layer，安全套接层）和 TLS（Transport Layer Security，传输层安全）协议把通信内容加密后经网络隧道传输。

TRACE

追踪路径

服务器会将通信路径返回给客户端。

发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器就会减 1，当数值为 0 时就停止传输。

通常不会使用 TRACE，并且它容易受到 XST 攻击（Cross-Site Tracing，跨站追踪）。

三、HTTP 状态码

服务器返回的 响应报文 中第一行为状态行，包含了状态码以及原因短语，用来告知客户端请求的结果。

状态码 类别 原因短语

1XX Informational（信息性状态码） 接收的请求正在处理

2XX Success（成功状态码） 请求正常处理完毕

3XX Redirection（重定向状态码） 需要进行附加操作以完成请求

4XX Client Error（客户端错误状态码） 服务器无法处理请求

5XX Server Error（服务器错误状态码） 服务器处理请求出错

1XX 信息

100 Continue ：表明到目前为止都很正常，客户端可以继续发送请求或者忽略这个响应。

2XX 成功

200 OK

204 No Content ：请求已经成功处理，但是返回的响应报文不包含实体的主体部分。一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。

206 Partial Content ：表示客户端进行了范围请求，响应报文包含由 Content-Range 指定范围的实体内容。

3XX 重定向

301 Moved Permanently ：永久性重定向

302 Found ：临时性重定向

303 See Other ：和 302 有着相同的功能，但是 303 明确要求客户端应该采用 GET 方法获取资源。

注：虽然 HTTP 协议规定 301、302 状态下重定向时不允许把 POST 方法改成 GET 方法，但是大多数浏览器都会在 301、302 和 303 状态下的重定向把 POST 方法改成 GET 方法。

304 Not Modified ：如果请求报文首部包含一些条件，例如：If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since，如果不满足条件，则服务器会返回 304 状态码。

307 Temporary Redirect ：临时重定向，与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 POST 方法改成 GET 方法。

4XX 客户端错误

400 Bad Request ：请求报文中存在语法错误。

401 Unauthorized ：该状态码表示发送的请求需要有认证信息（BASIC 认证、DIGEST 认证）。如果之前已进行过一次请求，则表示用户认证失败。

403 Forbidden ：请求被拒绝。

404 Not Found

5XX 服务器错误

500 Internal Server Error ：服务器正在执行请求时发生错误。

503 Service Unavailable ：服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

四、HTTP 首部

有 4 种类型的首部字段：通用首部字段、请求首部字段、响应首部字段和实体首部字段。

各种首部字段及其含义如下（不需要全记，仅供查阅）。

通用首部字段

首部字段名 说明

Cache-Control 控制缓存的行为

Connection 控制不再转发给代理的首部字段、管理持久连接

Date 创建报文的日期时间

Pragma 报文指令

Trailer 报文末端的首部一览

Transfer-Encoding 指定报文主体的传输编码方式

Upgrade 升级为其他协议

Via 代理服务器的相关信息

Warning 错误通知

请求首部字段

首部字段名 说明

Accept 用户代理可处理的媒体类型

Accept-Charset 优先的字符集

Accept-Encoding 优先的内容编码

Accept-Language 优先的语言（自然语言）

Authorization Web 认证信息

Expect 期待服务器的特定行为

From 用户的电子邮箱地址

Host 请求资源所在服务器

If-Match 比较实体标记（ETag）

If-Modified-Since 比较资源的更新时间

If-None-Match 比较实体标记（与 If-Match 相反）

If-Range 资源未更新时发送实体 Byte 的范围请求

If-Unmodified-Since 比较资源的更新时间（与 If-Modified-Since 相反）

Max-Forwards 最大传输逐跳数

Proxy-Authorization 代理服务器要求客户端的认证信息

Range 实体的字节范围请求

Referer 对请求中 URI 的原始获取方

TE 传输编码的优先级

User-Agent HTTP 客户端程序的信息

响应首部字段

首部字段名 说明

Accept-Ranges 是否接受字节范围请求

Age 推算资源创建经过时间

ETag 资源的匹配信息

Location 令客户端重定向至指定 URI

Proxy-Authenticate 代理服务器对客户端的认证信息

Retry-After 对再次发起请求的时机要求

Server HTTP 服务器的安装信息

Vary 代理服务器缓存的管理信息

WWW-Authenticate 服务器对客户端的认证信息

实体首部字段

首部字段名 说明

Allow 资源可支持的 HTTP 方法

Content-Encoding 实体主体适用的编码方式

Content-Language 实体主体的自然语言

Content-Length 实体主体的大小

Content-Location 替代对应资源的 URI

Content-MD5 实体主体的报文摘要

Content-Range 实体主体的位置范围

Content-Type 实体主体的媒体类型

Expires 实体主体过期的日期时间

Last-Modified 资源的最后修改日期时间

五、具体应用

连接管理

1. 短连接与长连接

当浏览器访问一个包含多张图片的 HTML 页面时，除了请求访问 HTML 页面资源，还会请求图片资源。如果每进行一次 HTTP 通信就要新建一个 TCP 连接，那么开销会很大。

长连接只需要建立一次 TCP 连接就能进行多次 HTTP 通信。

从 HTTP/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，使用 Connection : close；

在 HTTP/1.1 之前默认是短连接的，如果需要使用长连接，则使用 Connection : Keep-Alive。

2. 流水线

默认情况下，HTTP 请求是按顺序发出的，下一个请求只有在当前请求收到响应之后才会被发出。由于会受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间。

流水线是在同一条长连接上发出连续的请求，而不用等待响应返回，这样可以避免连接延迟。

Cookie

HTTP 协议是无状态的，主要是为了让 HTTP 协议尽可能简单，使得它能够处理大量事务。HTTP/1.1 引入 Cookie 来保存状态信息。

Cookie 是服务器发送到用户浏览器并保存在本地的一小块数据，它会在浏览器之后向同一服务器再次发起请求时被携带上，用于告知服务端两个请求是否来自同一浏览器。由于之后每次请求都会需要携带 Cookie 数据，因此会带来额外的性能开销（尤其是在移动环境下）。

Cookie 曾一度用于客户端数据的存储，因为当时并没有其它合适的存储办法而作为唯一的存储手段，但现在随着现代浏览器开始支持各种各样的存储方式，Cookie 渐渐被淘汰。新的浏览器 API 已经允许开发者直接将数据存储到本地，如使用 Web storage API（本地存储和会话存储）或 IndexedDB。

1. 用途

会话状态管理（如用户登录状态、购物车、游戏分数或其它需要记录的信息）

个性化设置（如用户自定义设置、主题等）

浏览器行为跟踪（如跟踪分析用户行为等）

2. 创建过程

服务器发送的响应报文包含 Set-Cookie 首部字段，客户端得到响应报文后把 Cookie 内容保存到浏览器中。

客户端之后对同一个服务器发送请求时，会从浏览器中取出 Cookie 信息并通过 Cookie 请求首部字段发送给服务器。

3. 分类

会话期 Cookie：浏览器关闭之后它会被自动删除，也就是说它仅在会话期内有效。

持久性 Cookie：指定一个特定的过期时间（Expires）或有效期（max-age）之后就成为了持久性的 Cookie。

4. 作用域

Domain 标识指定了哪些主机可以接受 Cookie。如果不指定，默认为当前文档的主机（不包含子域名）。如果指定了 Domain，则一般包含子域名。例如，如果设置 Domain=mozilla.org，则 Cookie 也包含在子域名中（如 developer.mozilla.org）。

Path 标识指定了主机下的哪些路径可以接受 Cookie（该 URL 路径必须存在于请求 URL 中）。以字符 %x2F ("/") 作为路径分隔符，子路径也会被匹配。

5. JavaScript

通过 document.cookie 属性可创建新的 Cookie，也可通过该属性访问非 HttpOnly 标记的 Cookie。

6. HttpOnly

标记为 HttpOnly 的 Cookie 不能被 JavaScript 脚本调用。跨站脚本攻击 (XSS) 常常使用 JavaScript 的 document.cookie API 窃取用户的 Cookie 信息，因此使用 HttpOnly 标记可以在一定程度上避免 XSS 攻击。

7. Secure

标记为 Secure 的 Cookie 只能通过被 HTTPS 协议加密过的请求发送给服务端。但即便设置了 Secure 标记，敏感信息也不应该通过 Cookie 传输，因为 Cookie 有其固有的不安全性，Secure 标记也无法提供确实的安全保障。

8. Session

除了可以将用户信息通过 Cookie 存储在用户浏览器中，也可以利用 Session 存储在服务器端，存储在服务器端的信息更加安全。

Session 可以存储在服务器上的文件、数据库或者内存中。也可以将 Session 存储在 Redis 这种内存型数据库中，效率会更高。

使用 Session 维护用户登录状态的过程如下：

用户进行登录时，用户提交包含用户名和密码的表单，放入 HTTP 请求报文中；

服务器验证该用户名和密码，如果正确则把用户信息存储到 Redis 中，它在 Redis 中的 Key 称为 Session ID；

服务器返回的响应报文的 Set-Cookie 首部字段包含了这个 Session ID，客户端收到响应报文之后将该 Cookie 值存入浏览器中；

客户端之后对同一个服务器进行请求时会包含该 Cookie 值，服务器收到之后提取出 Session ID，从 Redis 中取出用户信息，继续之前的业务操作。

应该注意 Session ID 的安全性问题，不能让它被恶意攻击者轻易获取，那么就不能产生一个容易被猜到的 Session ID 值。此外，还需要经常重新生成 Session ID。在对安全性要求极高的场景下，例如转账等操作，除了使用 Session 管理用户状态之外，还需要对用户进行重新验证，比如重新输入密码，或者使用短信验证码等方式。

9. 浏览器禁用 Cookie

此时无法使用 Cookie 来保存用户信息，只能使用 Session。除此之外，不能再将 Session ID 存放到 Cookie 中，而是使用 URL 重写技术，将 Session ID 作为 URL 的参数进行传递。

10. Cookie 与 Session 选择

Cookie 只能存储 ASCII 码字符串，而 Session 则可以存取任何类型的数据，因此在考虑数据复杂性时首选 Session；

Cookie 存储在浏览器中，容易被恶意查看。如果非要将一些隐私数据存在 Cookie 中，可以将 Cookie 值进行加密，然后在服务器进行解密；

对于大型网站，如果用户所有的信息都存储在 Session 中，那么开销是非常大的，因此不建议将所有的用户信息都存储到 Session 中。

缓存

1. 优点

缓解服务器压力；

降低客户端获取资源的延迟：缓存通常位于内存中，读取缓存的速度更快。并且缓存在地理位置上也有可能比源服务器来得近，例如浏览器缓存。

2. 实现方法

让代理服务器进行缓存；

让客户端浏览器进行缓存。

3. Cache-Control

HTTP/1.1 通过 Cache-Control 首部字段来控制缓存。

3.1 禁止进行缓存

no-store 指令规定不能对请求或响应的任何一部分进行缓存。

3.2 强制确认缓存

no-cache 指令规定缓存服务器需要先向源服务器验证缓存资源的有效性，只有当缓存资源有效才将能使用该缓存对客户端的请求进行响应。