物联网中间件

第五章 WEB 基础——HTTP、HTTPS、BASE64、URL

Java RMI所依赖的通信协议为：

- A　JRMP
- B　IIOP
- C　GIOP
- D　HTTP

Web Service

- Web Service = Http + （XML or JSON）
  - 服务交互时通过http协议传输，传输数据遵循XML、JSON等通用数据标识语言

不同的Web Services技术

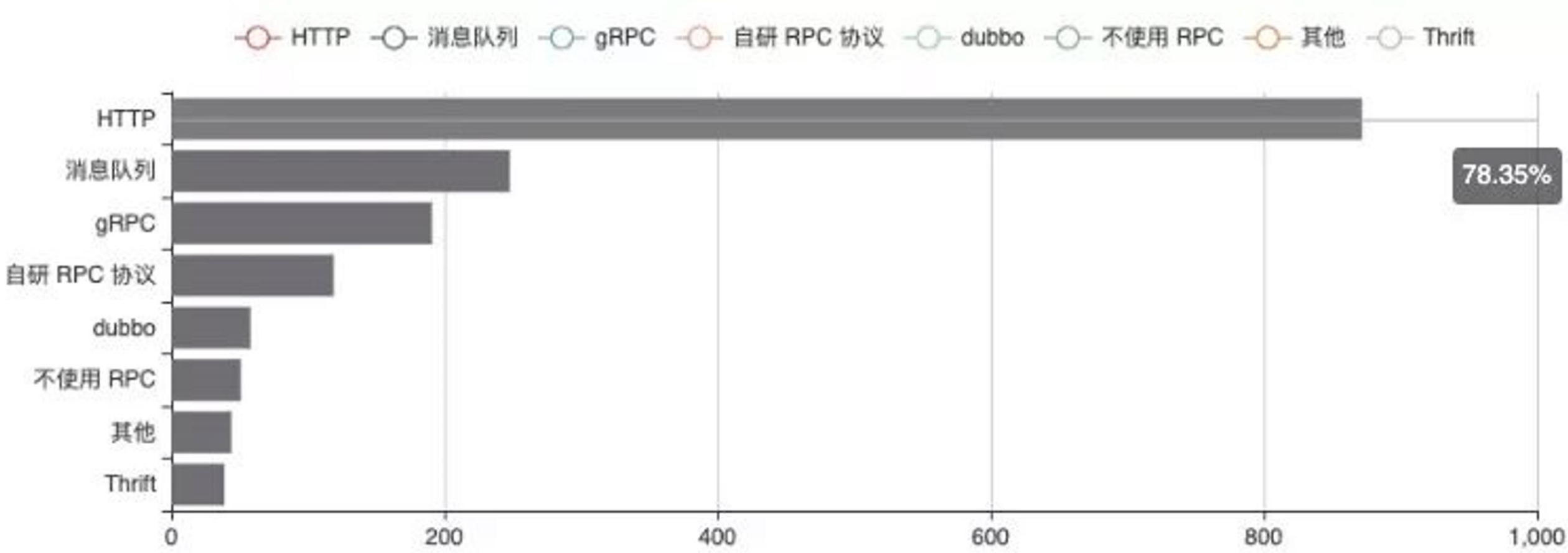
- 两大类Web Services技术
  - Non-RESTful
  - RESTful

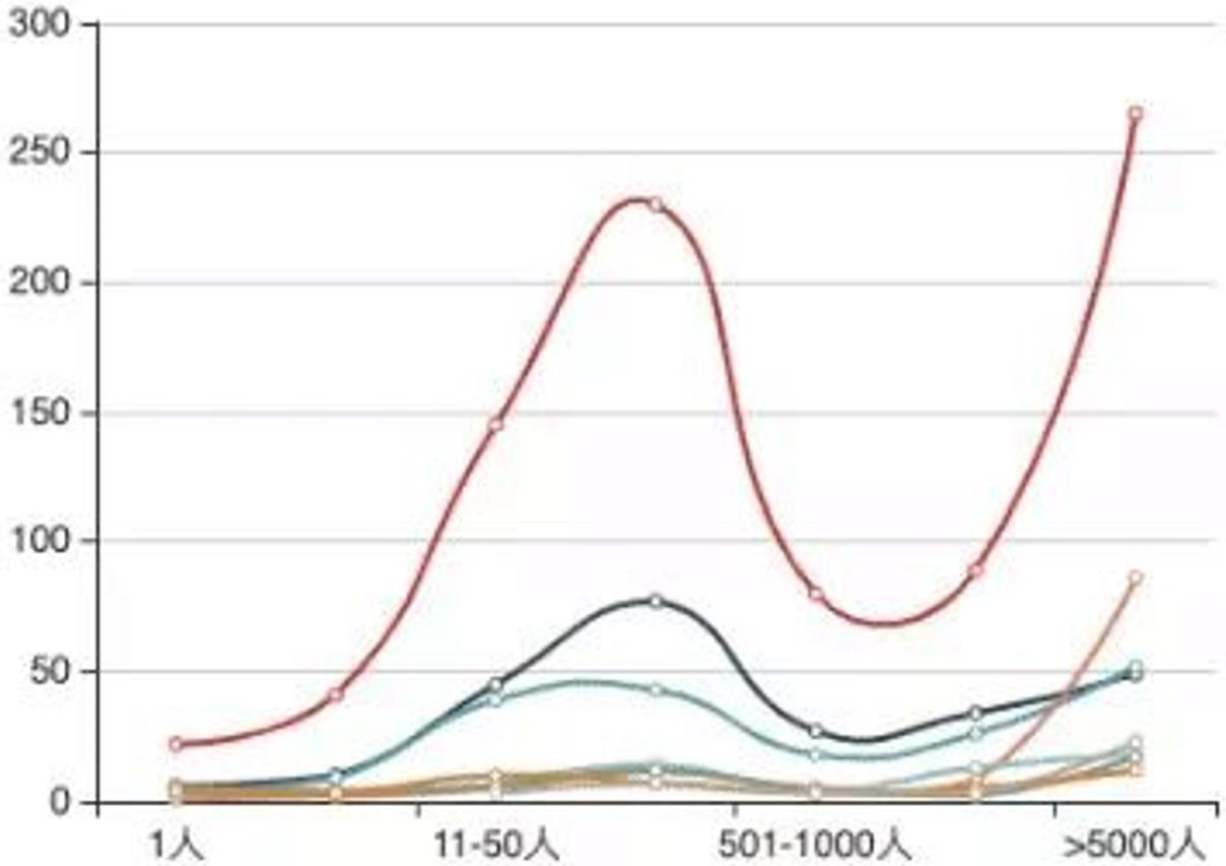
- 每一类Web Services技术之间是可以相互通信的，比如一个基于RESTful的技术可以调用另一个基于RESTful的技术。
- 不同类别的Web Services技术之间是不能相互通信的，比如一个基于non-RESTful的技术不可以调用一个基于RESTful的技术。

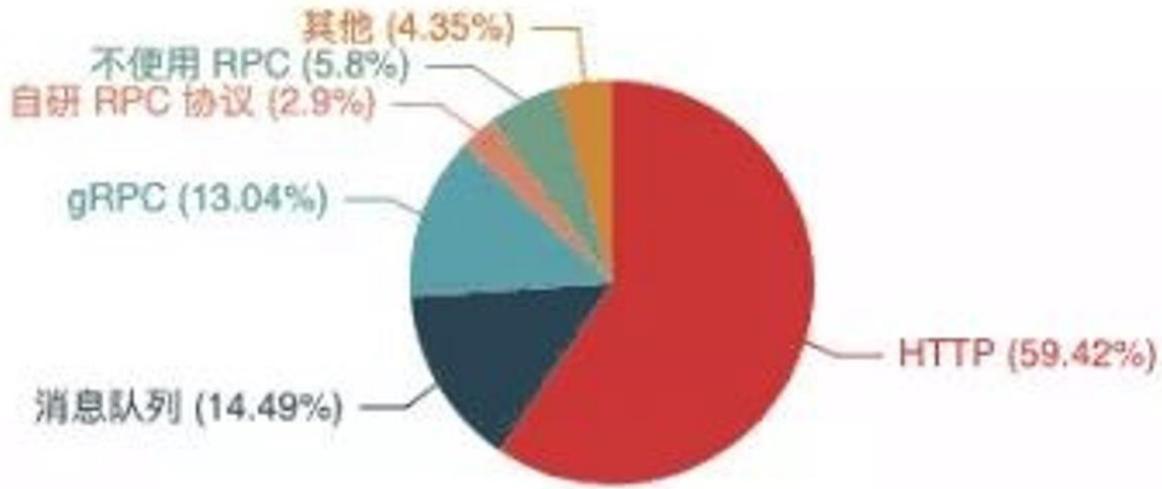
RPC

- **2020 年 Node.js 开发者调查报告**
  - https://zhuanlan.zhihu.com/p/131377284

- HTTP 是最常见的 RPC 方式，此外使用最多的是消息队列。
- 中型公司使用消息队列的比率较高。
- 大公司使用自研 RPC 协议的比率较高。


HTTP
消息队列
gRPC
自研 RPC 协议
dubbo
不使用 RPC
其他
Thrift
78.35%
0
200
400
600
800
1,000





阿里技术

不同规模公司的 RPC 方式

参考材料

- 课本 chapter 4、chapter 5

Web基础

- 1969年Internet 的前身ARPANET建立，1989年World Wide Wed（简称Web）建立
- Web运行在Internet之上，使用Internet进行通信
- 简单来说World Wide Wed由一系列通过超链接连接的文档组成，每个文档都有一个URL
- 通常情况下我们使用HTTP协议访问Internet上的文档
- Web上的文档通常使用HTML格式
- 数据在通过HTTP协议传输时，通常使用XML格式或者JSON格式

HTTP 简介

- HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

- HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

HTTP 工作原理

- HTTP协议工作于客户端-服务端架构上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。
- Web服务器根据接收到的请求后，向客户端发送响应信息。
- HTTP默认端口号为80，但是你也可以改为8080或者其他端口。

HTTP 特点

- HTTP是无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。（HTTP 0.9，HTTP 1.0）
- HTTP是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过HTTP发送。客户端以及服务器指定使用适合的MIME-type内容类型。
- HTTP是无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

HTTP 工作原理

- HTTP是基于客户端/服务端（C/S）的架构模型，通过一个可靠的连接交换信息，是一个无状态的请求/响应协议。
- 一个HTTP"客户端"是一个应用程序（Web浏览器或其他任何客户端），通过连接到服务器达到向服务器发送一个或多个HTTP的请求的目的。
- 一个HTTP"服务器"同样也是一个应用程序（通常是一个Web服务，如Apache Web服务器或IIS服务器等），通过接收客户端的请求并向客户端发送HTTP响应数据。
- HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）请求资源。一旦建立连接后，数据消息就通过类似Internet邮件所使用的格式[RFC5322]和多用途Internet邮件扩展（MIME）[RFC2045]来传送。

HTTP客户端请求消息

- 客户端发送一个HTTP请求到服务器

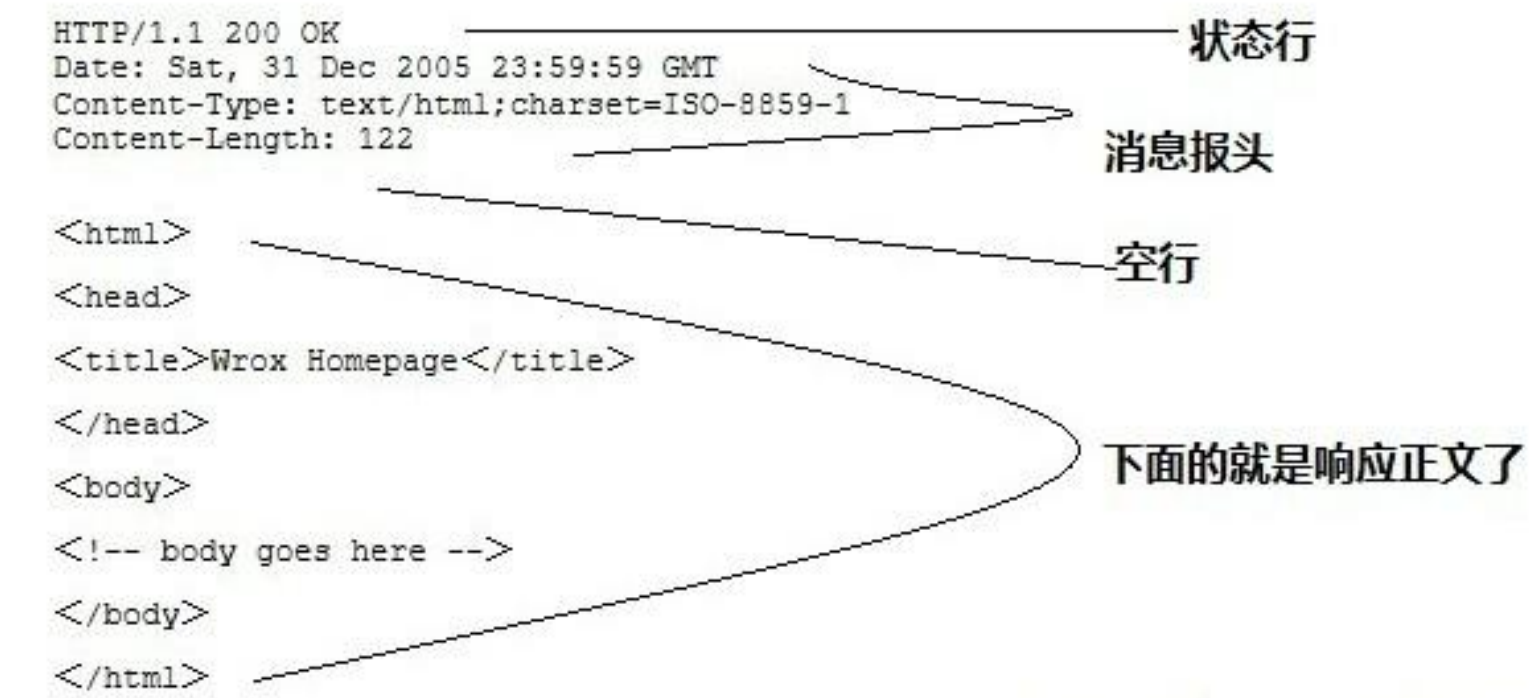
- 请求消息包括以下格式：请求行（request line）、请求头部（header）、空行和请求数据四个部分组成，下图给出了请求报文的一般格式。

HTTP客户端请求消息

HTTP服务器响应消息

- HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。

HTTP服务器响应消息



HTTP 请求方法（HTTP动词）

- 根据 HTTP 标准，HTTP 请求可以使用多种请求方法。
- HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD方法。
- HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

常见的HTTP方法

- GET：获取URI指定的资源。如果资源为文本直接返回，如果是CGI（Common Gateway Interface）那样的程序则返回执行后结果。
- POST：用于给服务器添加信息。
- PUT方法：传输文件到指定URI位置。不带验证机制，任何人均可上传存在安全问题，一般web禁止。采用REST（REpresentational State Transfer 表征状态转移）标准的网站可能会开放PUT。

常见的HTTP方法

- HEAD方法：与GET相似，但不获取报文主体，仅用来确认URI有效性以及资源更新日期。
- DELETE方法：删除URI指定的文件，与PUT相反
- OPTIONS方法：询问支持的方法，例如：OPTIONS * HTTP/1.1 HOST：baidu.com 来获取HOST指定服务器支持的方法。

HTTP状态码

- HTTP状态码的英文为HTTP Status Code。

- 当浏览者访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接收并显示网页前，此网页所在的服务器会返回一个包含HTTP状态码的信息头（server header）用以响应浏览器的请求。

- 状态码的职责是描述返回的请求结果。借助状态码，用户可以知道服务器端是正常处理了请求，还是出现了错误。

常见的HTTP状态码

- 200 - 请求成功
- 301 - 资源（网页等）被永久转移到其它URL
- 404 - 请求的资源（网页等）不存在
- 500 - 内部服务器错误

HTTP协议几个版本的比较

- HTTP 0.9
- HTTP 1.0
- HTTP 1.1
- HTTP 2.0

HTTP 0.9

- 1990年HTTP协议面世

- HTTP 0.9是第一个版本的HTTP协议，已过时。它的组成极其简单，只允许客户端发送GET这一种请求，且不支持请求头。由于没有协议头，造成了HTTP 0.9协议只支持一种内容，即纯文本。不过网页仍然支持用HTML语言格式化，同时无法插入图片。

- HTTP 0.9具有典型的无状态性，每个事务独立进行处理，事务结束时就释放这个连接

HTTP 0.9

- 一次HTTP 0.9的传输首先要建立一个由客户端到Web服务器的TCP连接，由客户端发起一个请求，然后由Web服务器返回页面内容，然后连接会关闭。如果请求的页面不存在，也不会返回任何错误码。

HTTP 1.0

- 1996年发布HTTP/1.0 规范。
- 请求与响应支持头域
- 响应对象以一个响应状态行开始
- 响应对象不只限于超文本
- 开始支持客户端通过POST方法向Web服务器提交数据，支持GET、HEAD、POST方法

HTTP 1.0

- 默认使用的是短连接: 每一个请求建立一个TCP连接，请求完成后立马断开连接。这将会导致:
  - 连接无法复用：连接无法复用会导致每次请求都经历三次握手和慢启动。三次握手在高延迟的场景下影响较明显，慢启动则对文件类请求影响较大。

- 缺少强制的 Host 首部

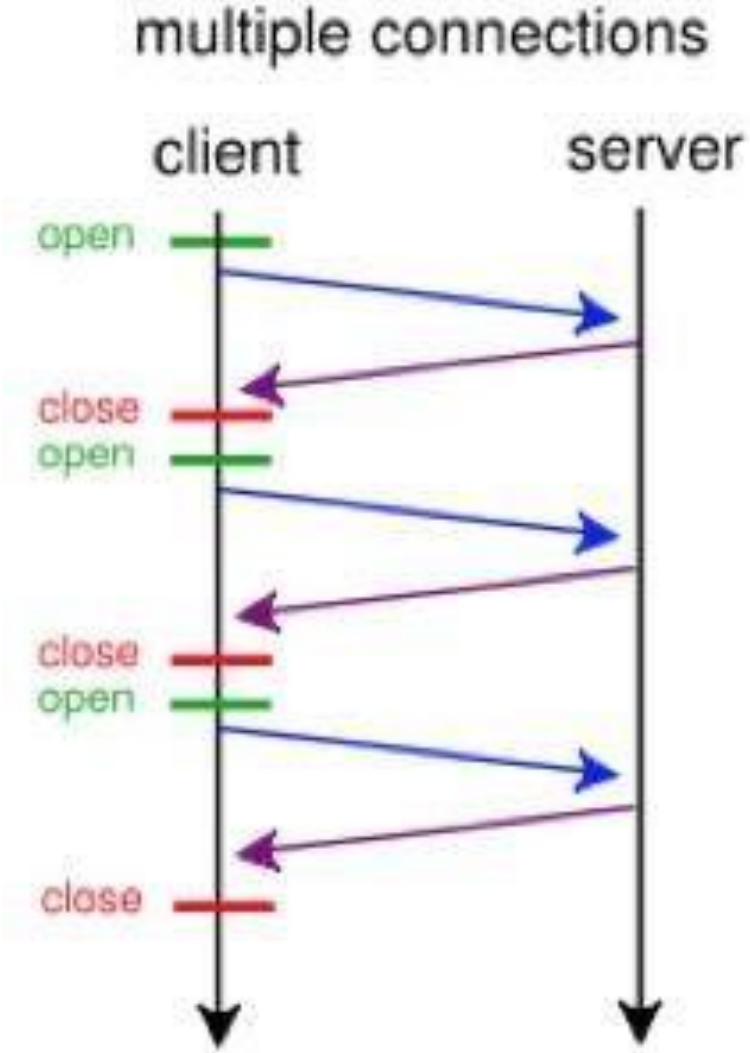
HTTP 1.1

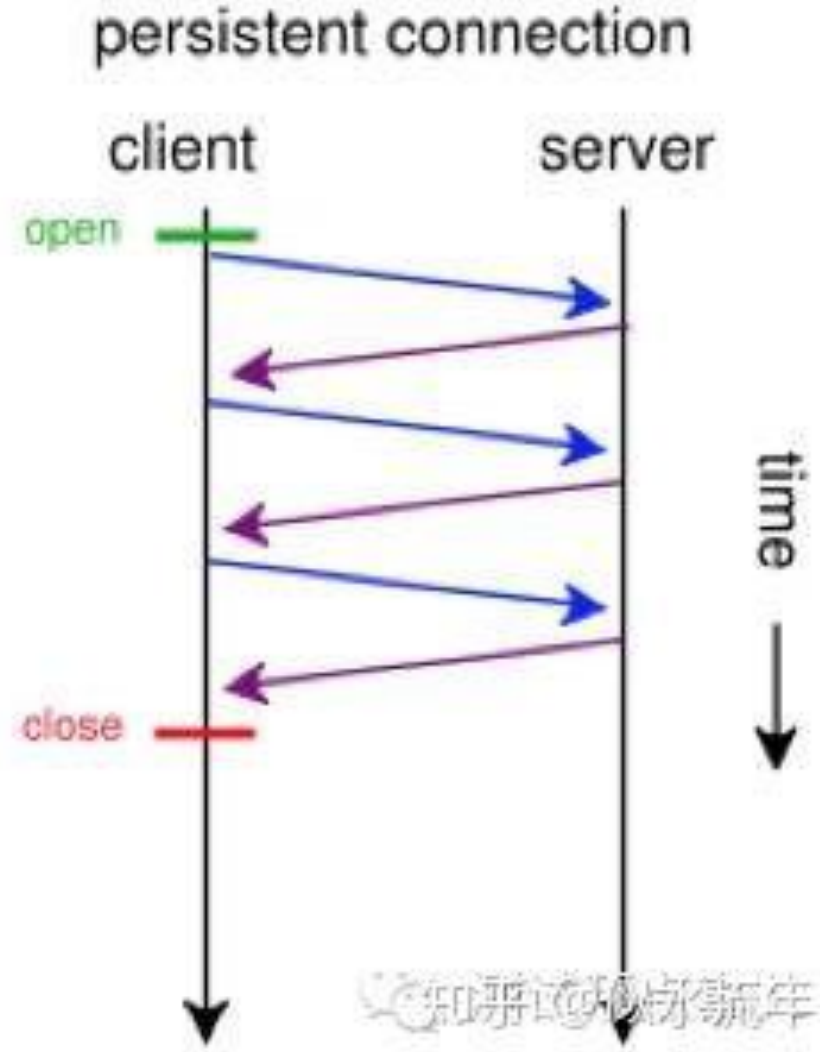
- 1997年发布了HTTP1.1版本，它修复了之前提到的 1.0 版本的大量问题。HTTP1.1是目前使用最广泛的协议版本，也是目前主流的HTTP协议版本，

- HTTP 1.1引入了许多关键性能优化：keepalive连接，chunked编码传输，字节范围请求，请求流水线等

Persistent Connection（keepalive连接）

- 允许HTTP设备在事务处理结束之后将TCP连接保持在打开的状态，以便未来的HTTP请求重用现在的连接，直到客户端或服务器端决定将其关闭为止。在HTTP1.0中使用长连接需要添加请求头Connection: Keep-Alive，而在HTTP 1.1 所有的连接默认都是长连接，除非特殊声明不支持（ HTTP请求报文首部加上Connection: close ）


multiple connections
client
server
open
close
open
close
open
close


persistent connection
client
server
open
close
time

HTTP 1.1还新增了如下特性

- 新增了一批Request method：HTTP1.1增加了OPTIONS,PUT, DELETE, TRACE, CONNECT方法
- chunked编码传输：该编码将实体分块传送并逐块标明长度。
- 字节范围请求：HTTP1.1支持传送内容的一部分。
- 请求消息和响应消息都支持Host首部：强制客户端提供Host首部，使得虚拟主机托管成为可能（一个IP提供多个服务器）

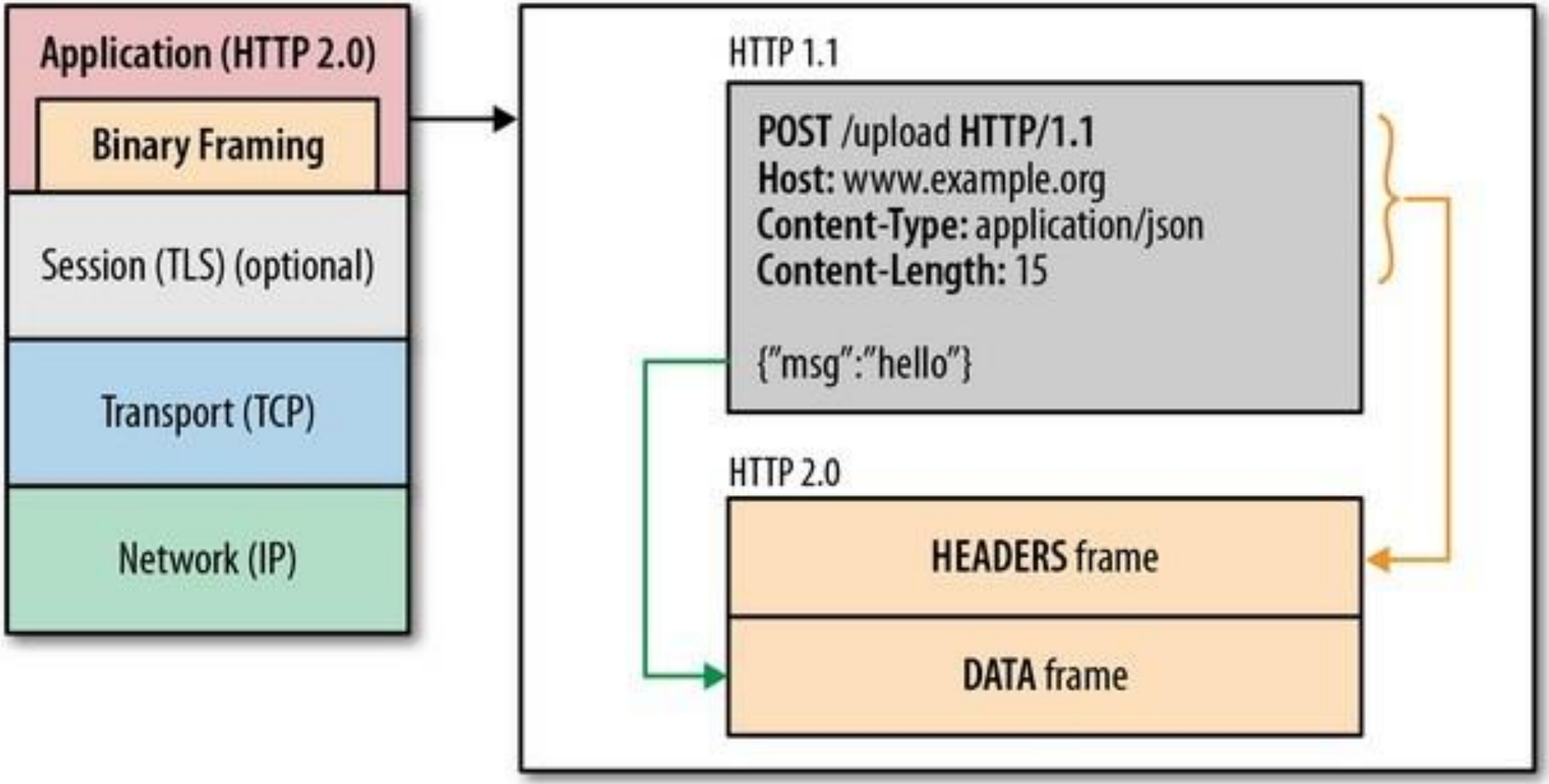
HTTP 2.0

- HTTP/2 官网
  - https://http2.github.io
- 2015年发布了HTTP2.0版本
- 目前HTTP/2.0（简称h2）已经在广泛使用（截止2018年8月根据Alexa流行度排名的头部1千万网站中，h2占比约29%）

二进制分帧

- http1.x一直都是plain text, 优点是便于阅读和debug。但是，现在很多都用https，SSL也把plain text变成了二进制，那这个优点也没了。

- 应用层(HTTP/2)和传输层(TCP or UDP)之间增加一个二进制分帧层
- 在二进制分帧层中， HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame）,并对它们采用二进制格式的编码， 其中 HTTP1.x 的首部信息会被封装到 HEADER frame，而相应的 Request Body 则封装到 DATA frame 里面。


Application (HTTP 2.0)
Binary Framing
Session (TLS) (optional)
Transport (TCP)
Network (IP)
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

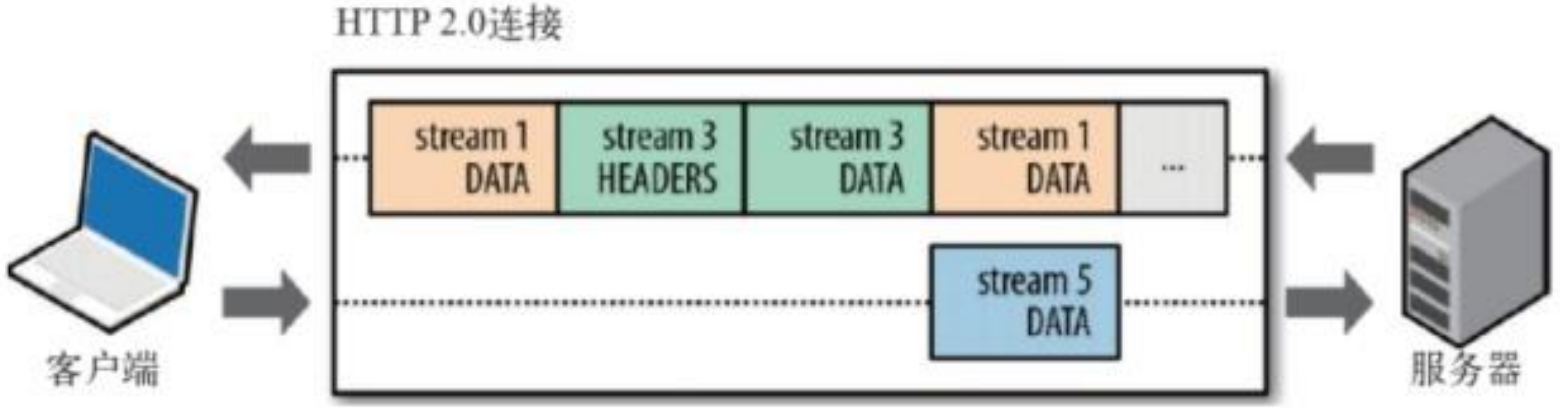
多路复用 (Multiplexing)

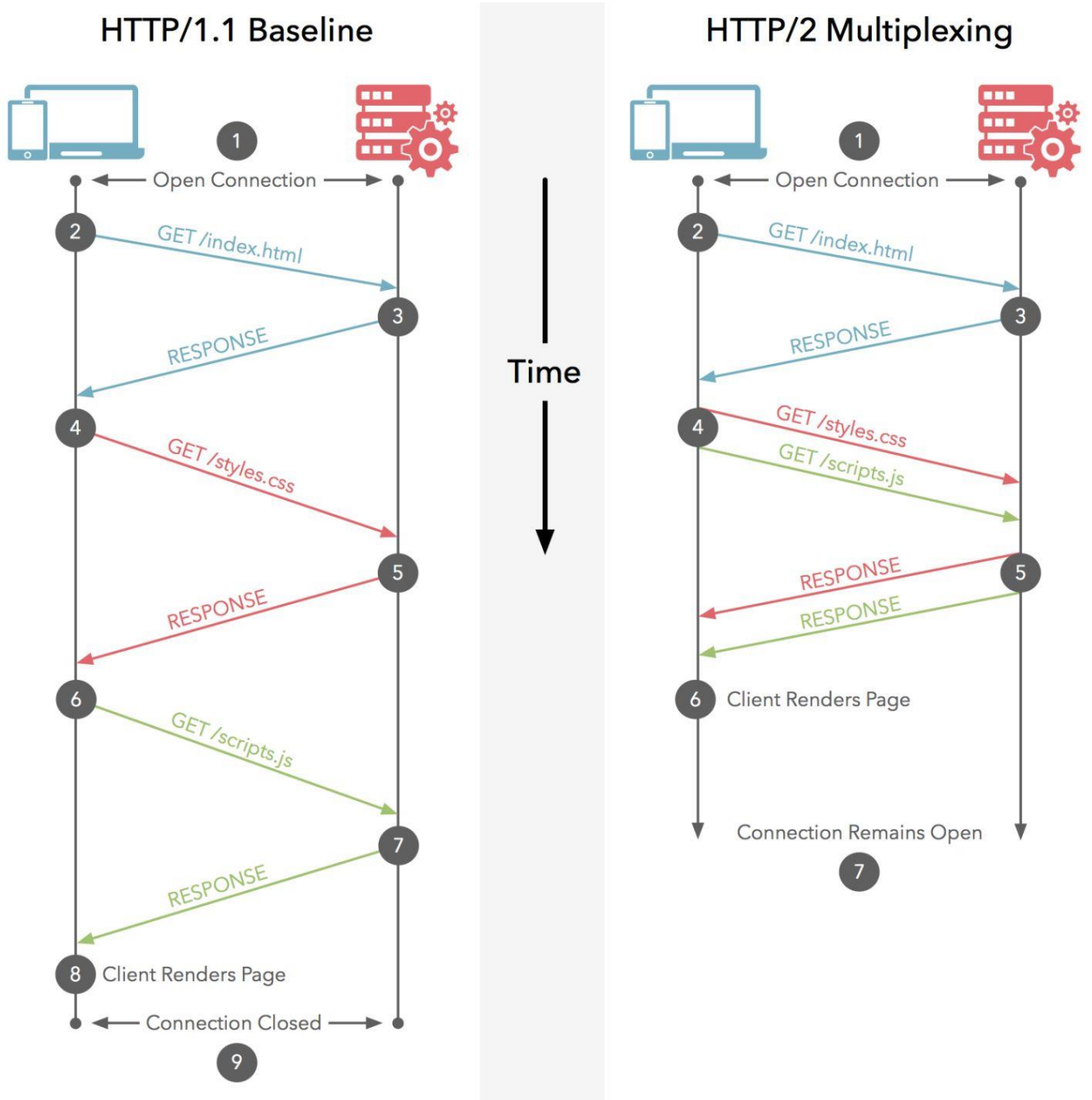
- HTTP/1.1 协议中，发送一次请求时，需要**等待服务端响应了**才可以继续发送请求。可能会出现**阻塞**的情况。从专业的名词上说这种情况，叫做**线头阻塞**（Head of line blocking）简称：HOLB。

- HTTP/2 的多路复用(Multiplexing) 则允许同时通过单一的 HTTP/2 连接发起多重的**请求-响应**消息。

多路复用 (Multiplexing)

- http2连接可以承载数十或数百个流的复用，多路复用意味着来自很多流的数据包能够混合在一起通过同样连接传输。当到达终点时，再根据不同帧首部的流标识符重新连接将不同的数据流进行组装。




HTTP/1.1 Baseline
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
5
RESPONSE
6
GET /scripts.js
7
RESPONSE
8
Client Renders Page
Connection Closed
9
Time
HTTP/2 Multiplexing
1
Open Connection
2
GET /index.html
3
RESPONSE
4
GET /styles.css
GET /scripts.js
5
RESPONSE
RESPONSE
6
Client Renders Page
Connection Remains Open
7

首部压缩（Header Compression）

- HTTP2用HPACK压缩来压缩头部，减少报文大小
- HPACK 算法：专门为首部压缩而设计

服务端推动Sever Push

- 这个功能通常被称作“缓存推送”。主要的思想是：当一个客户端请求资源X，而服务器知道它很可能也需要资源Z的情况下，服务器可以在客户端发送请求前，主动将资源Z推送给客户端。

- 这个功能帮助客户端将Z放进缓存以备将来之需。

HTTP 3.0

- https://http3.net/

- https://quicwg.org/base-drafts/draft-ietf-quic-http.html

HTTPS

- HTTPS 协议（HyperText Transfer Protocol over Secure Socket Layer）：一般理解为HTTP+SSL/TLS，通过 SSL证书来验证服务器的身份，并为浏览器和服务器之间的通信进行加密。

- SSL（Secure Socket Layer，安全套接字层）：1994年为 Netscape 所研发，SSL 协议位于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。

TLS

- TLS（Transport Layer Security，传输层安全）：其前身是 SSL，它最初的几个版本（SSL 1.0、SSL 2.0、SSL 3.0）由网景公司开发，1999年从 3.1 开始被 IETF 标准化并改名，发展至今已经有 TLS 1.0、TLS 1.1、TLS 1.2 三个版本。SSL3.0和TLS1.0由于存在安全漏洞，已经很少被使用到。目前使用最广泛的是TLS 1.1、TLS 1.2。

SSL发展史

- 1994年NetSpace公司设计SSL协议（Secure Sockets Layout）1.0版本，但未发布。
- 1995年NetSpace发布SSL/2.0版本，很快发现有严重漏洞
- 1996年发布SSL/3.0版本，得到大规模应用
- 1999年，发布了SSL升级版TLS/1.0版本
- 2006年和2008年，发布了TLS/1.1版本和TLS/1.2版本

https效果

- Confidentiality：你浏览的页面的内容如果被人中途看见，将会是一团乱码。
- Intergrity：你浏览的页面就是你想浏览的，不会被黑客在中途修改，网站收到的数据包也是你最初发的那个，不会把你的数据给换掉
- Authentication：你连接的是你连接的网站，而不是什么人在中途伪造了一个网站给你（专业上叫Man In The Middle Attack）

加密算法

- 对称加密
  - 有流式、分组两种，加密和解密都是使用的同一个密钥。
  - 例如：DES、AES-GCM、ChaCha20-Poly1305等

非对称加密

- 加密使用的密钥和解密使用的密钥是不相同的，分别称为：公钥、私钥，公钥和算法都是公开的，私钥是保密的。

- 非对称加密算法性能较低，但是安全性超强，由于其加密特性，非对称加密算法能加密的数据长度也是有限的。
- 例如：RSA、DSA、ECDSA、 DH、ECDHE

非对称加密

- 每个用户都有两把钥匙，一把公钥一把私钥。公钥是对外发布的，所有人都看的到所有人的公钥，私钥是自己保存，每个人都只知道自己的私钥而不知道别人的。

- 用该用户的公钥加密后只能该用户的私钥才能解密。这种情况下，公钥是用来加密信息的，确保只有特定的人（用谁的公钥就是谁）才能解密该信息。

非对称加密

- 还有第二种情况，公钥是用来解密信息的，确保让别人知道这条信息是真的由我发布的，是完整正确的。接收者由此可知这条信息确实来自于拥有私钥的某人，这被称作数字签名，公钥的形式就是数字证书。

RSA

- RSA(RSA (algorithm) , Ronald Rivest , Adi Shamir and Leonard Adleman)以三位创始人的名字命名
- RSA加密：非对称密钥，公开密钥算法
- RSA加密利用了单向函数正向求解很简单，反向求解很复杂的特性。

RSA算法原理

- 1.对两个质数相乘容易，而将其合数分解很难的这个特点进行的加密算法。

- n=p1*p2，已知p1、p2求n简单，已知n求p1、p2困难。

RSA加密算法的组成

- 原文(Message)：需要加密的信息，可以是数字、文字、视频、音频等，用 M 表示。
- 密文(Ciphertext)：加密后得到的信息，用 C 表示。
- 公钥(Public Key)和私钥(Private Key)，用 PU 和 PR 表示。
- 加密算法(Encryption)：若 $E(x)$ 为加密算法，加密过程可以理解为 $C = E(M)$ 根据原文和加密算法得到密文。
- 解密算法(Decryption)：若 $D(x)$ 为解密算法，解密过程可以理解为 $M = D(C)$ 根据密文和解密算法得到原文。

RSA加密算法的PU、PR生成步骤

1. 随机选择两个不相同的素数 p, q 。

- 随机数生成算法：生成随机数（其实是伪随机数）
  - 时间复杂度：多项式级（相对于输出规模）

- Miller-Rabin测试：测试一个数是否是质数。（有概率误判，伪素数可能蒙混过关。单次测试的可信度还不是非常高。需要用不同的参数进行多次独立测试，让误概率小到可以忽略不计。）
  - 时间复杂度：多项式级（相对于输入）

2. 将 p, q 相乘，记为 $n = p \times q$ 。

3. 计算 n 的欧拉函数 $\varphi(n)$ ，欧拉函数证明，当 p, q 为不相同的素数时，

$$\varphi(n) = (p-1)(q-1) \text{ 。}$$

欧拉函数

- 欧拉函数——φ()
- φ(n)表示：小于n的正整数中与n互质的数的数目。（互质表示公因数为1）
- 比如想要知道φ(10)的话，我们就可以看[1, 10)中和10互质的整数，也就是1、3、7、9这四个数。（2、4、6、8和10有公因数2，而5和10有公因数10）。所以φ(10)=4。

- 比如想要知道φ(21)的话，我们就可以看[1, 21)中和21互质的整数，也就是1、2、4、5、8、10、11、13、16、17、19、20这12个数。（3、6、9、12、15、18和21有公因数3，而7、14和21有公因数7）。所以φ(21)=12。

- 比如10=2*5，2和5是两个质数，所以φ(10)=(2-1)*(5-1)=4。
- 比如21=3*7，3和7是两个质数，所以φ(21)=(3-1)*(7-1)=12。

4. 随机选择一个整数 e ，满足两个条件： $\varphi(n)$ 与 e 互质，且 $1 < e < \varphi(n)$ 。
5. 计算 e 对于 $\varphi(n)$ 的模反元素 d ，也就是说找到一个 d 满足 $ed \equiv 1 \, mod \, \varphi(n)$ 。这个式子等价于 $ed - 1 = k\varphi(n)$ ，实际上就是对于方程 $ed - k\varphi(n) = 1$ 求 (d, k) 的整数解。这个方程可以用扩展欧几里得算法求解。
6. 最终把 (e, n) 封装成公钥， (d, n) 封装成私钥。

同余式

- 比如说，23÷7=3……2
- 再比如，65÷7=9……2
- 我们发现，这两个数除以7都余2，
- 可以这样写：　23 ≡ 65（mod 7）
- 当然也可以这样：
- 　23 ≡ 2（mod 7）
- 65 ≡ 2（mod 7）
- 如果a=b+km的话，a ≡ b（mod m）

同余式性质:

- 同余式可以互相加:
- 若a ≡ b（mod m）、c ≡ d（mod m）
- 则a+c ≡ b+d（mod m）

- 同余式可以互相乘:
- 若a ≡ b（mod m）、c ≡ d（mod m）
- 则a*c ≡ b*d（mod m）

欧拉公式

若a与n互质，则$a^{\varphi(n)} \equiv 1 \pmod{n}$

- **对于一个与n互质的数a：**

因为 $a^{\varphi(n)} \equiv 1 \pmod{n}$

所以 $a^{k\varphi(n)} \equiv 1 \pmod{n}$

所以 $a^{k\varphi(n)+1} \equiv a \pmod{n}$

所以 $a^{ed} \equiv a \pmod{n}$

所以，若 $c \equiv a^e \pmod{n}$，则 $c^d \equiv a^{ed} \equiv a \pmod{n}$

- 公钥：（n, e）

$$c \equiv a^e \pmod{n}$$ (快速幂算法，多项式时间)

- 私钥：（n, d）

$$a \equiv c^d \pmod{n}$$

回顾上面的加密过程，我们用到了六个变量： $p, q, n, \varphi(n), e, d$ ，其中只有公钥 (e, n) 是公开的。想要破解密文，只要知道私钥 (d, n) ，计算 $M = C^d mod\, n$ 就可以破解RSA算法。

1. 很明显，因为公钥中已知 n ，只要计算出 d ，就能得到私钥。
2. $ed \equiv 1(mod\, \varphi(n))$ ，需要知道 e 和 $\varphi(n)$ 的值来求出 d 。因为在公钥中已知 e ，所以只要求出 $\varphi(n)$ 的值。
3. $\varphi(n) = (p-1)(q-1)$ ，要求出 $\varphi(n)$ 的值，需要求出 p, q 的值。
4. $n = p \times q$ ，想要求出 p, q 的值，必须对n做因数分解。

- **分解质因数**
- 时间复杂度：**指数级**（相对于输入规模）

哈希算法

- 哈希函数（Hash Function），也称为散列函数，给定一个输入x，它会算出相应的输出H(x)。哈希函数的主要特征是：
  - 输入x可以是任意长度的字符串
  - 输出结果即H(x)的长度是固定的
  - 计算 H(x) 的过程是高效的（对于长度为 n 的字符串 x，计算出 H(x) 的时间复杂度应为 O(n) ）

哈希函数特点

- **抗碰撞能力：**对于任意两个不同的数据块，其hash值相同的可能性极小

- 即不会出现输入 x≠y ，但是H(x)=H(y) 的情况，其实这个特点在理论上并不成立，比如目前比特币使用的 SHA256 算法，会有 2^256 种输出，如果我们进行 2^256 + 1 次输入，那么必然会产生一次碰撞，事实上，通过 理论证明 ，通过 2^130 次输入就会有99%的可能性发生一次碰撞，不过即使如此，即便是人类制造的所有计算机自宇宙诞生开始一直运算到今天，发生一次碰撞的几率也是极其微小的。

哈希函数特点

- **隐匿性**：也就是说，对于一个给定的输出结果 H(x)，想要逆推出输入 x，在计算上是不可能的。如果想要得到 H(x) 的可能的原输入，不存在比穷举更好的方法。

- **抗篡改能力**：对于一个数据块，哪怕只改动其一个比特位，其hash值的改动也会非常大。

MD4

- **消息摘要算法版本4（Message Digest Algorithm 4）**
- MD4是麻省理工学院教授 Ronald Rivest 于1990年设计出来的算法。其摘要长度为128位，一般用32位的十六进制来表示。

- 2004年8月清华大学教授王小云，指出在计算MD4时可能发生杂凑冲撞。不久之后，Dobbertin 等人发现了MD4在计算过程中第一步和第三步中的漏洞，并向大家演示了如何利用一部普通电脑在几分钟内找到MD4中的冲突，毫无疑问，MD4就此被淘汰掉了。

MD5

- **消息摘要算法版本5 （Message Digest Algorithm 5）**
- 1991年，Rivest 开发出技术上更为趋近成熟的MD5算法，它在MD4的基础上增加了"安全-带子"（safety-belts）的概念。虽然MD5 比 MD4 复杂度大一些，但却更为安全。

SHA系列

- SHA的全称是Secure Hash Algorithm(安全hash算法)，SHA系列有五个算法，分别是 SHA-1、SHA-224、SHA-256、SHA-384，和 SHA-512，由美国国家安全局（NSA）所设计，并由美国国家标准与技术研究院（NIST）发布，是美国的政府标准。后四者有时并称为 SHA-2。 SHA系列算法在许多安全协定中广为使用，包括 TLS/SSL 等，是 MD5 的后继者。

SHA-1

- 最初该算法于1993年发布，称做安全散列标准 (Secure Hash Standard)，最初这个版本被称为"SHA-0"，它在发布之后很快就被NSA撤回，因为有很大的安全缺陷，之后在1995年发布了修订版本，也就是SHA-1。

- SHA-0 和 SHA-1 会从一个最大 2^64 位（2147483648GB）元的讯息中产生一串 160 位元的摘要

SHA-1

- 2017年，谷歌发布了最新的研究成功，宣布攻破了SHA-1，并详细描述了成功的SHA1碰撞攻击方式，使用这种方式，可以在亚马逊的云计算平台上，耗时10天左右创建出SHA-1碰撞，并且成本可以控制在11万美元以内。

- 即使如此，对于单台机器来说攻击的成本依然很高，发生一次SHA-1碰撞需要超过9,223,372,036,854,775,808个SHA1计算，这需要使用你的机器进行6500年计算。

SHA2

- SHA2包括了SHA-224、SHA-256、SHA-384，和SHA-512，这几个函数都将讯息对应到更长的讯息摘要，以它们的摘要长度（以位元计算）加在原名后面来命名，也就是说SHA-256会产生256位长度摘要。

- SHA-2相对来说是安全的，至今尚未出现对SHA-2有效的攻击!

数字签名

- 数字签名
  - 签名就是在信息的后面再加上一段内容（信息经过hash后的值），可以证明信息没有被修改过。
  - hash值一般都会加密后（也就是签名）再和信息一起发送，以保证这个hash值不被修改


客户端
服务器
① Client Hello
② Server Hello
③ [用对称加密，对称加密算法和对称密钥]
公钥加密
公钥
私钥
④ [好的]
对称密钥加密
对称密钥
对称密钥
⑤ [请求内容]
对称密钥加密
⑥ [响应内容]
对称密钥加密

对称加密，非对称加密两者结合

- （1）第 ③ 步时，客户端说：（咱们后续回话采用对称加密吧，这是对称加密的算法和对称密钥）这段话用公钥进行加密，然后传给服务器

- （2）服务器收到信息后，用私钥解密，提取出对称加密算法和对称密钥后，服务器说：（好的）对称密钥加密

- （3）后续两者之间信息的传输就可以使用对称加密的方式了

获取公钥与确认服务器身份

- （1）客户端如何获得公钥
- （2）如何确认服务器是真实的而不是黑客

SSL 证书


客户端
服务器
① Client Hello
② Server Hello 这是我的 SSL证书
公钥
③ [用对称加密，对称加密算法和对称密钥]
公钥加密
私钥
④ [好的]
对称密钥加密
对称密钥
⑤ [请求内容]
对称密钥加密
对称密钥
⑥ [响应内容]
对称密钥加密

SSL 证书中包含的具体内容

- （1）证书的发布机构CA
- （2）证书的有效期
- （3）公钥
- （4）证书所有者
- （5）签名

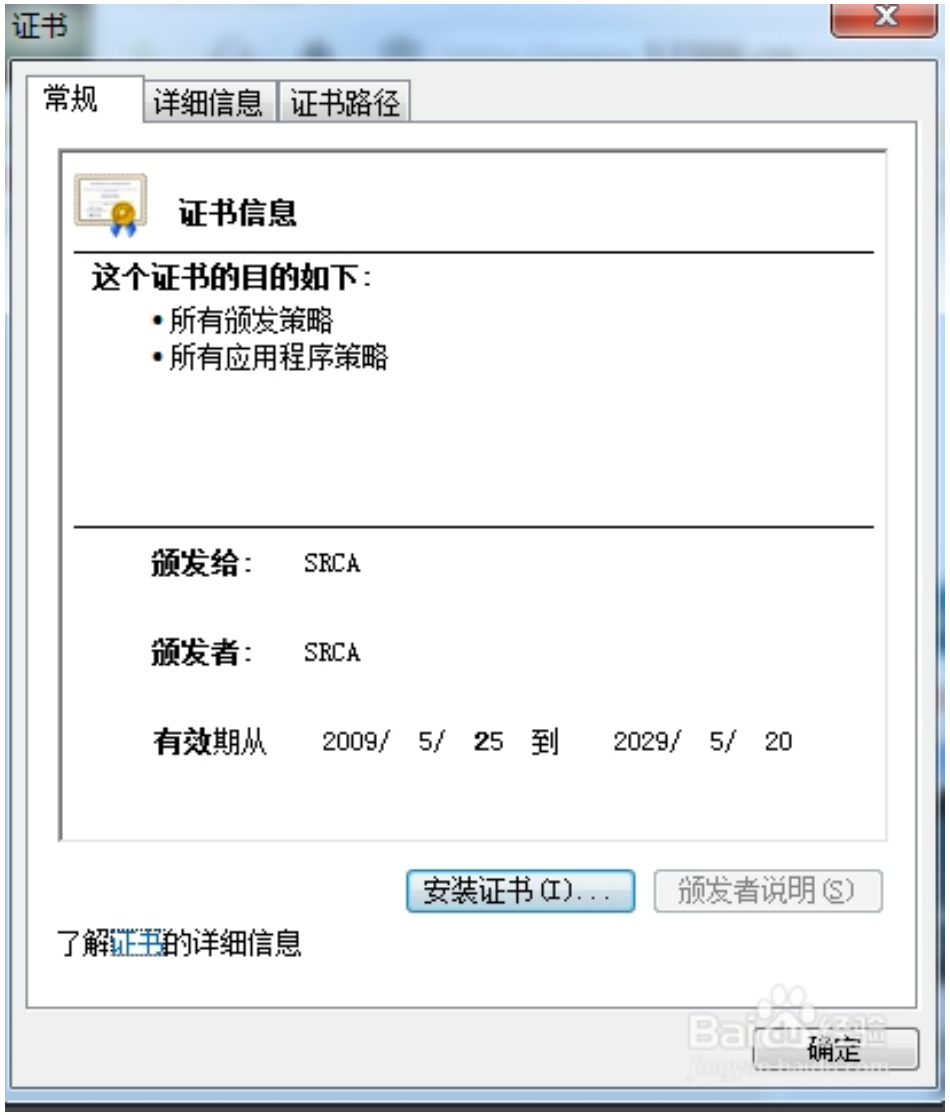
客户端校验SSL 证书

- （1）首先浏览器读取证书中的证书所有者、有效期等信息进行一一校验
- （2）浏览器开始查找操作系统中已内置的受信任的证书发布机构CA，与服务器发来的证书中的颁发者CA比对，用于校验证书是否为合法机构颁发
- （3）如果找不到，浏览器就会报错，说明服务器发来的证书是不可信任的。

客户端校验SSL 证书

- （4）如果找到，那么浏览器就会从操作系统中取出 颁发者CA 的公钥，然后对服务器发来的证书里面的签名进行解密
- （5）浏览器使用相同的hash算法计算出服务器发来的证书的hash值，将这个计算的hash值与证书中签名做对比
- （6）对比结果一致，则证明服务器发来的证书合法，没有被冒充
- （7）此时浏览器就可以读取证书中的公钥，用于后续加密了


中国铁路客户服务中心
www.12306.cn是中国铁路客户服务中心唯一网站。截止目前，没有授权其他网站开展类似服务内容，敬请广大用户注意
2014年4月13日 星期日
首 页
客运服务
货运服务
行包服务
车站引导
铁路常识
站车风采
客户信箱
站车风采
北京西站
更多>>>
旅客服务质量调查问卷
最新动态
为保障您顺畅购票，请下载安装根证书。
关于2014年上半年京沪、京广高铁部分G字头动车组列车商务、特等、一...
(2014-03-27)
关于2014年短途卧铺优惠有关事宜的公告
(2014-03-05)
关于网站对用户身份信息进行核验的公告
(2014-02-23)
(2014-02-23)
(2014-01-07)
(2014-04-12)
(2014-04-10)
更多>>>
新建下载任务
网址：
http://www.12306.cn/mormhweb/ggxxfw/wbyyzj/201106/sr
名称：
srca12306.zip
154 KB
下载到：
C:\Users\HOME\Desktop
浏览
类型： WinRAR ZIP 压缩文件
C盘剩余空间:46.9 GB
使用迅雷下载
直接打开
下载
取消
我要发货
中国铁路货运电子商务平台
货运主要营业站受理服务电话
货运运费查询
货运业务咨询
新版售票
点击进入>>
网上购票用户注册
购 票
退 票
余 票 查 询
旅客列车时刻表查询
网上购票常见问题
铁路常识
货运办理常见问题
在互联网购买了儿童票，如何在售票窗口换取纸质车票？
注册用户时，系统提示身份信息重复怎么办？
购买实名制车票后丢失了怎么办？
网上购票由于安全警告无法登录问题说明
铁路货运
规范性文件
搜 索
相关链接
中央政府门户网站
外交部
发展改革委
教育部


证书
常规
详细信息
证书路径
证书信息
这个证书的目的如下:
• 所有颁发策略
• 所有应用程序策略
颁发给: SRCA
颁发者: SRCA
有效期从 2009/ 5/ 25 到 2029/ 5/ 20
安装证书(I)...
颁发者说明(S)
了解证书的详细信息
确定

HTTPS传输数据的流程

- 首先客户端通过URL访问服务器建立SSL连接。
- 服务端收到客户端请求后，会将网站支持的证书信息（证书中包含公钥）传送一份给客户端。
- 客户端和服务器开始协商SSL连接的安全等级，也就是信息加密的等级。
- 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。
- 服务器利用自己的私钥解密出会话密钥。服务器利用会话密钥加密与客户端之间的通信。

HTTPS的缺点

- HTTPS协议多次握手，导致页面的加载时间延长近50%;
- HTTPS连接不如HTTP高效，会增加数据开销和功耗;
- 申请SSL证书需要钱，功能越强大的证书费用越高。
- SSL涉及到的安全算法会消耗 CPU 资源，对服务器资源消耗较大

总结HTTPS和HTTP的区别

- HTTPS是HTTP协议的安全版本，HTTP协议的数据传输是明文的，是不安全的，HTTPS使用了SSL/TLS协议进行了加密处理。

- http和https使用连接方式不同，默认端口也不一样，http是80，https是443。

BASE 64 编码

- Base 64可以把所有的二进制数据都转换成ASCII码可打印字符串的形式。以便在只支持文本的环境中也能够顺利地传输二进制数据。
  - Email中附件只能使用文本格式
  - Xml、JSON

Base64的编码原理

- Base64编码的核心原理是将二进制数据进行分组，每24Bit(3字节)为一个大组，再把一个大组的数据分成4个6Bit的小分组。

- 由于6Bit数据只能表示64个不同的字符(2^6=64)，所以这也是Base64的名字由来。

- 这64个字符分别对应ASCII码表中的'A'-'Z'，'a'-'z'，'0''9'，'+'和'/'。他们的对应关系是由Base64字符集决定的。

Base64的编码原理

- 因为小分组中的6Bit数据表示起来并不方便，所以要把每个小分组进行高位补零操作，这样每个小分组就构成了一个8Bit(字节)的数据。

- 在补零操作完成后接下来的工作就简单多了，那就是将小分组的内容作为Base64字符集的下标，然后一一替换成对应的ASCII字符。编码工作完成。

TABLE 4.2
The Base 64 Value to Character Encodings

Value	Character	Value	Character	Value	Character	Value	Character
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Base64的编码实例

- 一个3字节的二进制数为例
  - 01010111 01101000 01101111
- 3个字节连接在一起共有24位
  - 010101110110100001101111
- 把这24位4个六位的块
  - 010101 110110 100001 101111

Base64的编码实例

- 每个6位的数据块都可以用一个10进制数标识
  - 010101 = 21 base 10
  - 110110 = 54 base 10
  - 100001 = 33 base 10
  - 101111 = 47 base 10

- 根据前面的对应表，我们就能够得到数据Base64表示
  - V2hv

Base64的解码原理

- Base64解码的工作原理也非常的简单，只要操作方式和加密步骤相反即可。首先将Base64编码根据其对应的字符集转换成下标，这就是补完零后的8Bit(一字节)数据。
- 既然有补零操作那自然会有去零操作了，我们要将这些8Bit数据的最高位上的两个0抹去形成6Bit数据，这也就是前面我们提到过的小分组。
- 最后就是将每4个6Bit数据进行合并形成24Bit的大分组，然后将这些大分组按照每组8Bit进行拆分就会得到3个8Bit的数据，这写8Bit数据就是编码前的数据了。解码工作完成。

Base64的编码原理

- Base64编码前是无法保证准备编码的字符串长度是3的倍数，所以为了让编码能够顺利进行就必须在获取编码字符串的同时判断字符串的长度是否是3的倍数。
- 如果是3的倍数编码就可以正常进行，如果不是那么就要进行额外的操作——补零，就是要在不足3的倍数的字符串末尾用0x00进行填充。
  这样就是解决了字符串长度不足的问题了，但是同时也引进了另一个新的问题，那就是末尾补充上的0在进行Base64字符集替换的时候会与字符集中的'A'字符发生冲突。

Base64的编码原理

- 因为字符集中的下标0对应的字符是'A'，而末尾填充上的0x00在分组补零后同样是下标0x00，这样就无法分辨出到底是末尾填充的0x00还是二进制数据中的0x00。
- 为了解决这个问题我们就必须引入Base64字符集外的新字符来区分末尾补充上的0x00，这就是'='字符不在Base64字符集中，但是也出现在Base64编码的原因了，'='字符在一个Base64编码的末尾中最多会出现两个，如果不符合这以规则那么这个Base64就可能被人做了手脚。

Base64的编码实例

- 我们的数据最后还剩下一个字节需要用BASE64编码
  - 01010111
- 把这个字节通过末位填充0的方式扩展为3个字节
  - 01010111 00000000 00000000
- 转换为6位的数据块
  - 010101 110000 000000 000000
- 根据前面的对应表，我们就能够得到数据Base64表示
  - Vw==

URL编码

- URL是为了统一的命名网络中的资源

- URL有一些基本的特性:
  - URL是可移植的。（所有的网络协议都可以使用URL）
  - URL的完整性。（不能丢失数据）
  - URL的可阅读性。（希望人能阅读）

URL编码

- Urlencoding（URL编码）： 只有字母和数字[0-9a-zA-Z]、一些特殊符号“$-_.+!*'(),”[不包括双引号]、以及某些保留字，才可以不经过编码直接用于URL。

- 由于 URL 常常会包含 ASCII 集合之外的字符，URL 必须转换为有效的 ASCII 格式。URL 编码使用 "%" 其后跟随两位的十六进制数来替换非 ASCII 字符。

URL编码

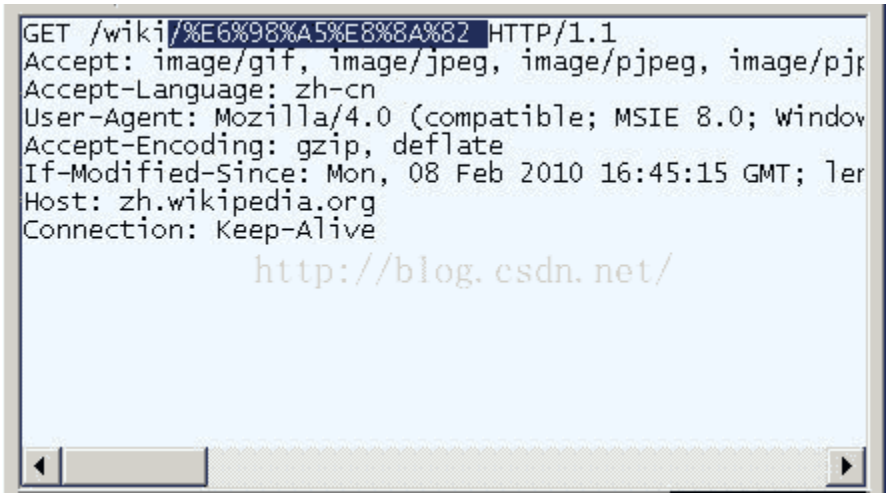
- 特殊符号在URL中是不能直接传递的，如果要在URL中传递这些特殊符号，那么就要使用他们的编码了。编码的格式为：%加字符的ASCII码，即一个百分号%，后面跟对应字符的ASCII（16进制）码值。例如空格的编码值是"%20"

URL编码例子

- 网址中含有汉字:
  - 网址http://zh.wikipedia.org/wiki/春节

URL编码例子

- 实际查询的网址是http://zh.wikipedia.org/wiki/%E6%98%A5%E8%8A%82



- “春”和“节”的utf-8编码分别是“E6 98 A5”和“E8 8A 82”，因此，“%E6%98%A5%E8%8A%82”就是按照顺序，在每个字节前加上%而得到的

URL编码例子

- 部分URL特殊符号及编码

序号	特殊字符	含义	十六进制值
1.	+	URL 中+号表示空格	%2B
2.	空格	URL中的空格可以用+号或者编码	%20
3.	/	分隔目录和子目录	%2F
4.	?	分隔实际的 URL 和参数	%3F
5.	%	指定特殊字符	%25
6.	#	表示书签	%23
7.	&	URL 中指定的参数间的分隔符	%26
8.	=	URL 中指定参数的值	%3D

URI,URN,URL

- URI: Uniform Resource Identifier (统一资源标识符): 用一个紧凑的字符串用来标示抽象或物理资源
  - 也就是说,URI只是规定如何标识资源, 没有规定如何获取资源 也就是 **what the resource is but I don't care how to get the resource**

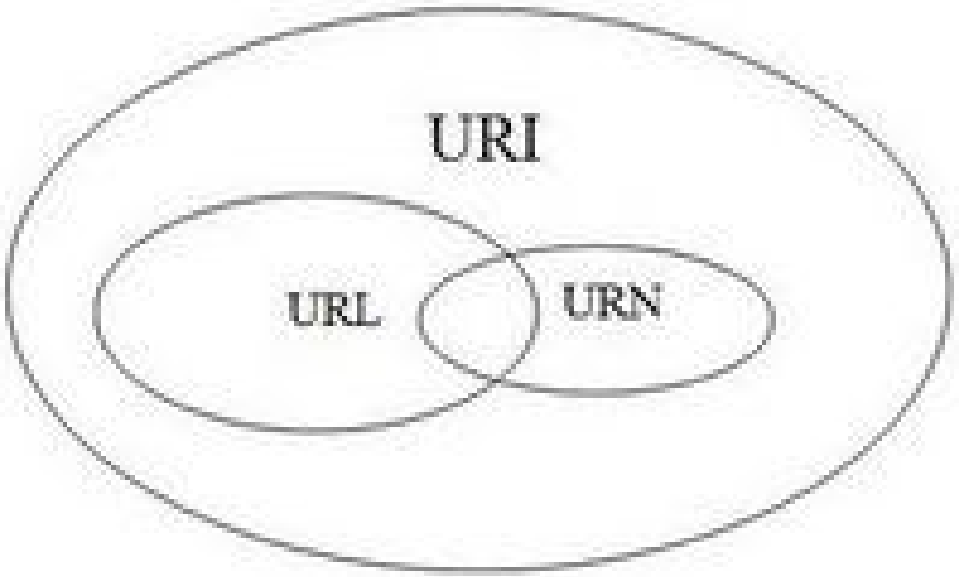
- URL:Uniform Resource Locator (统一资源定位符): URL是URI最常见的表现形式.它明确说明如何从一个精准、固定的位置获取资源
  - URL不但规定了如何标识资源, 还规定了如何获取资源..也就是**what the resource is && how to get the resource**

URI,URN,URL

- URN: Uniform Resource Name (统一资源名称):URN作为特定内容的唯一名称使用,与目前的资源所在地无关, 使用这些与位置无关的URN,就可以将资源到处搬移
  - 通过URN,可以使用相同一个名字通过多种网络访问协议来访问资源，也就是**I know the resource name and get the resource, but you don't need to know how can i do.**
  - URN一般都是**urn: 作为开头**

URI,URN,URL

- URI是URL与URN的父类,URI是抽象的表现, **所有的URL与URN都是URI**, 但是URI不一定就是URL或者URN



举例说明

- **URI:**
  以下(URL|URN)都是URI
- **URL:**
  - http://example.com/mypage.html
  - ftp://example.com/download.zip
- **URN:**
  - urn:isbn:0451450523 书的ISBN编码
  - magnet:?xt=urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJ
  - urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJ 文件有唯一数字指纹

答疑