HTTP协议简介

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是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中，而且HTTP-NG(Next Generation of HTTP)的建议已经提出。

HTTP协议工作于客户端-服务端架构为上。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五大特点

1. 支持**客户/服务器**模式。
2. **简单快速**：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有`GET`、`HEAD`、`POST`。每种方法规定了客户与服务器联系的类型不同。由于`HTTP`协议简单，使得`HTTP`服务器的程序规模小，因而通信速度很快。
3. **灵活**：HTTP允许传输任意类型的数据对象。正在传输的类型由`Content-Type`加以标记。
4. **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。采用这种方式可以节省传输时间。早期这么做的原因是请求资源少，追求快。后来通过`Connection: Keep-Alive`实现长连接
5. **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之URL

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息

URL,全称是UniformResourceLocator，中文叫统一资源定位符,是互联网上用来标识某一处资源的地址。以下面这个URL为例，介绍下普通URL的各部分组成

https://www.xxx.com:8080/news/1.html?a=1&b=2#name

从上面的URL可以看出，一个完整的URL包括以下几部分：

1. **协议部分**：该URL的协议部分为“http：”，这代表网页使用的是HTTP协议。在Internet中可以使用多种协议，如HTTP，FTP等等本例中使用的是HTTP协议。在"HTTP"后面的“//”为分隔符
2. **域名部分**：该URL的域名部分为“www.aspxfans.com”。一个URL中，也可以使用IP地址作为域名使用
3. **端口部分**：跟在域名后面的是端口，域名和端口之间使用“:”作为分隔符。端口不是一个URL必须的部分，如果省略端口部分，将采用默认端口
4. **虚拟目录部**分：从域名后的第一个“/”开始到最后一个“/”为止，是虚拟目录部分。虚拟目录也不是一个URL必须的部分。本例中的虚拟目录是“/news/”
5. **文件名部分**：从域名后的最后一个“/”开始到“？”为止，是文件名部分，如果没有“?”，则是从域名后的最后一个“/”开始到“#”为止，是文件部分，如果没有“？”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分。本例中的文件名是“index.asp”。文件名部分也不是一个URL必须的部分，如果省略该部分，则使用默认的文件名
6. **锚部分**：从“#”开始到最后，都是锚部分。本例中的锚部分是“name”。锚部分也不是一个URL必须的部分
7. **参数部分**：从“？”开始到“#”为止之间的部分为参数部分，又称搜索部分、查询部分。本例中的参数部分为“boardID=5&ID=24618&page=1”。参数可以允许有多个参数，参数与参数之间用“&”作为分隔符

URL和URI的区别

URI，是uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。

Web上可用的每种资源如HTML文档、图像、视频片段、程序等都是一个来URI来定位的 URI一般由三部组成： ①访问资源的命名机制 ②存放资源的主机名 ③资源自身的名称，由路径表示，着重强调于资源。

URL是uniform resource locator，统一资源定位器，它是一种具体的URI，即URL可以用来标识一个资源，而且还指明了如何locate这个资源。

URL是Internet上用来描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上，特别是著名的Mosaic。 采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。URL一般由三部组成： ①协议(或称为服务方式) ②存有该资源的主机IP地址(有时也包括端口号) ③主机资源的具体地址。如目录和文件名等

URN，uniform resource name，统一资源命名，是通过名字来标识资源，比如 `mailto:java-net@java.mjj.com` **。**

URI是以一种抽象的，高层次概念定义统一资源标识，而URL和URN则是具体的资源标识的方式。URL和URN都是一种URI。笼统地说，每个 URL 都是 URI，但不一定每个 URI 都是 URL。这是因为 URI 还包括一个子类，即统一资源名称(URN)，它命名资源但不指定如何定位资源。上面的 mailto、news 和 isbn URI 都是 URN 的示例。

在Java的URI中，一个URI实例可以代表绝对的，也可以是相对的，只要它符合URI的语法规则。而URL类则不仅符合语义，还包含了定位该资源的信息，因此它不能是相对的。 在Java类库中，URI类不包含任何访问资源的方法，它唯一的作用就是解析。 相反的是，URL类可以打开一个到达资源的流。

HTTP之请求

由上图可以看到，http请求由请求行，消息报头，请求正文三部分构成。

HTTP请求状态行

请求行由请求`Method`，`URL`字段和`HTTP Version`三部分构成，总的来说请求行就是定义了本次请求的请求方式，请求的地址，以及所遵循的HTTP协议版本例如：

```
GET /example.html HTTP/1.1 (CRLF)
```

HTTP协议的方法有：

- `GET`： 请求**获取**Request-URI所标识的资源
- `POST`： 在Request-URI所标识的资源后**增加**新的数据
- `HEAD`： 请求获取由Request-URI所标识的资源的**响应消息报头**
- `PUT`： 请求服务器**存储或修改**一个资源，并用Request-URI作为其标识
- `DELETE`： 请求服务器**删除**Request-URI所标识的资源
- `TRACE`： 请求服务器回送收到的请求信息，主要用于**测试或诊断**
- `CONNECT`： 保留将来使用
- `OPTIONS`： 请求查询服务器的性能，或者查询与资源相关的选项和需求

HTTP请求头

消息报头由一系列的键值对组成，允许客户端向服务器端发送一些附加信息或者客户端自身的信息，主要包括：

Header	解释	示例
Accept	指定客户端能够接收的内容类型	Accept: text/plain, text/html
Accept-Charset	浏览器可以接受的字符编码集	Accept-Charset: iso-8859-5,utf-8
Accept-Encoding	指定浏览器可以支持的web服务器返回内容压缩编码类型	Accept-Encoding: compress, gzip
Accept-		

Language	浏览器可接受的语言	Accept-Language: en,zh
Accept-Ranges	可以请求网页实体的一个或者多个子范围字段	Accept-Ranges: bytes
Authorization	HTTP授权的授权证书类型	Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Cache-Control	指定请求和响应遵循的缓存机制	Cache-Control: no-cache
Connection	表示是否需要持久连接（HTTP 1.1默认进行持久连接）	Connection: close
Cookie	HTTP请求发送时，会把保存在该请求域名下的所有cookie值一起发送给web服务器	Cookie: $Version=1; Skin=new;
Content-Length	请求的内容长度	Content-Length: 348
Content-Type	请求的与实体对应的MIME信息	Content-Type: application/x-www-form-urlencoded
Date	请求发送的日期和时间	Date: Tue, 15 Nov 2010 08:12:31 GMT
Expect	请求的特定的服务器行为	Expect: 100-continue
From	发出请求的用户的Email	From: user@email.com
Host	指定请求的服务器的域名和端口号	Host: www.zcmhi.com
If-Match	只有请求内容与实体相匹配才有效	If-Match: “737060cd8c284d8af7ad3082f209582d”
If-Modified-Since	如果请求的部分在指定时间之后被修改则请求成功，未被修改则返回304代码	If-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT
If-None-	如果内容未改变返回304代码，参数为服务器先前发送的Etag，与服务器回应的Etag	If-None-Match: “737060cd8c284d8af7ad3

Match	比较判断是否改变	082f209582d”
If-Range	如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体。参数也为Etag	If-Range: “737060cd8c284d8af7ad3082f209582d”
If-Unmodified-Since	只在实体在指定时间之后未被修改才请求成功	If-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT
Max-Forwards	限制信息通过代理和网关传送的时间	Max-Forwards: 10
Pragma	用来包含实现特定的指令	Pragma: no-cache
Proxy-Authorization	连接到代理的授权证书	Proxy-Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Range	只请求实体的一部分，指定范围	Range: bytes=500-999
Referer	先前网页的地址，当前请求网页紧随其后，即来路	Referer: www.zcmhi.com/archives/71…
TE	客户端愿意接受的传输编码，并通知服务器接受接受尾加头信息	TE: trailers,deflate;q=0.5
Upgrade	向服务器指定某种传输协议以便服务器进行转换（如果支持）	Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
User-Agent	User-Agent的内容包含发出请求的用户信息	User-Agent: Mozilla/5.0 (Linux; X11)
Via	通知中间网关或代理服务器地址，通信协议	Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
Warning	关于消息实体的警告信息	Warn: 199

		Miscellaneous warning

HTTP请求正文

只有在发送 `POST` 请求时才会有请求正文， `GET` 方法并没有请求正文。

HTTP响应

与HTTP请求类似，先上一张图：

HTTP响应也由三部分组成，包括状态行，消息报头，响应正文。

HTTP响应状态行

状态行也由三部分组成，包括HTTP协议的版本，状态码，以及对状态码的文本描述。例如：

```
HTTP/1.1 200 OK  (CRLF)
```

HTTP响应状态码

状态代码有三位数字组成，第一个数字定义了响应的**类别**，且有五种可能取值：

- `1xx`：**指示信息** - 表示请求已接收，继续处理
- `2xx`：**成功** - 表示请求已被成功接收、理解、接受
- `3xx`：**重定向** - 要完成请求必须进行更进一步的操作
- `4xx`：**客户端错误** - 请求有语法错误或请求无法实现
- `5xx`：**服务器端错误** - 服务器未能实现合法的请求

常见状态代码、状态描述、说明：

- `200`：**OK** - 客户端请求成功
- `400`：**Bad Request** - 客户端请求有语法错误，不能被服务器所理解
- `401`：**Unauthorized** - 请求未经授权，这个状态代码必须和`WWW-Authenticate`报头域一起使用
- `403`：**Forbidden** - 服务器收到请求，但是拒绝提供服务
- `404`：**Not Found** - 请求资源不存在，eg：输入了错误的URL

- `500`： **Internal Server Error** - 服务器发生不可预期的错误
- `503`： **Server Unavailable** - 服务器当前不能处理客户端的请求，一段时间后,可能恢复正常

HTTP响应状态码说明

StatusCode	StatusCode语义	中文描述
100	Continue	继续。客户端应继续其请求
101	Switching Protocols	切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
200	OK	请求成功。一般用于GET与POST请求
201	Created	已创建。成功请求并创建了新的资源
202	Accepted	已接受。已经接受请求，但未处理完成
203	Non-Authoritative Information	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206	Partial Content	部分内容。服务器成功处理了部分GET请求

300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found 临时移动。	与301类似。但资源只是临时被移动。客户端应继续使用原有URI
303	See Other	查看其它地址。与301类似。使用GET和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理。所请求的资源必须通过代理访问
306	Unused	已经被废弃的HTTP状态码
307	Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
40	Not	

6	Acceptable	服务器无法根据客户端请求的内容特性完成请求
407	Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
408	Request Time-out	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的PUT请求是可能返回此代码，服务器处理请求时发生了冲突
410	Gone	客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
411	Length Required	服务器无法处理客户端发送的不带Content-Length的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
414	Request-URI Too Larg	请求的URI过长（URI通常为网址），服务器无法处理
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式
416	Requested range not satisfiable	客户端请求的范围无效
417	Expectation Failed	服务器无法满足Expect的请求头信息

500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	充当网关或代理的服务器，从远端服务器接收到了一个无效的请求
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

HTTP响应报文

HTTP和HTTPS

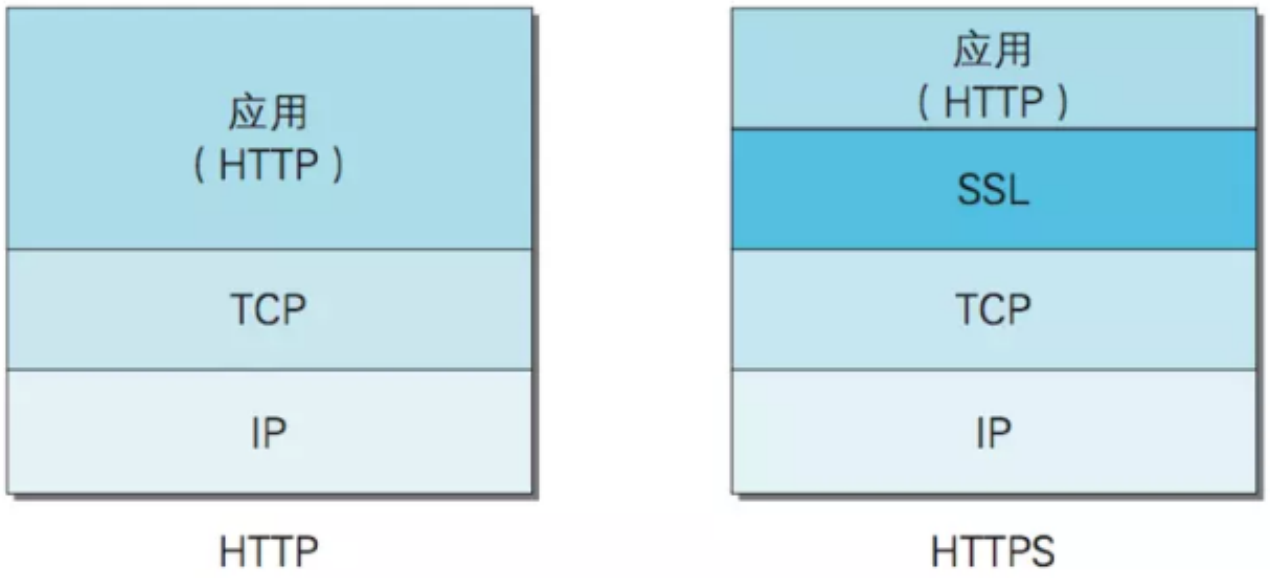
HTTP的不足

- 通信使用明文(不加密),内容可能会被窃听
- 不验证通信方的身份,因此有可能遭遇伪装
- 无法证明报文的完整性,所以有可能已遭篡改

HTTPS介绍

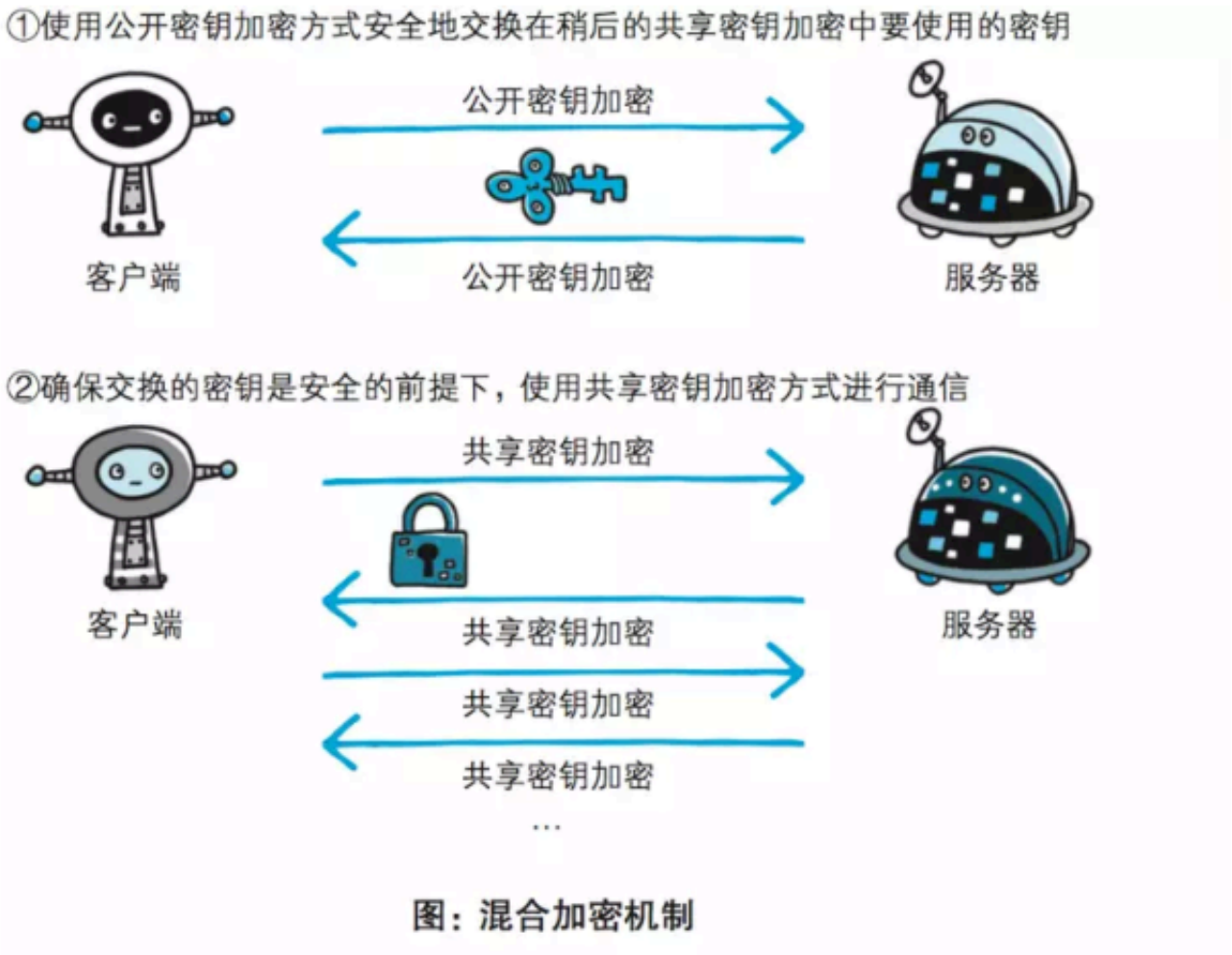
`HTTP` 协议中没有加密机制,但可以通 过和 `SSL`(Secure Socket Layer, **安全套接层**)或 `TLS`(Transport Layer Security, **安全层传输协议**)的组合使用,加密 `HTTP` 的通信内容。属于通信加密，即在整个通信线路中加密。

```
HTTP + 加密 + 认证 + 完整性保护 = HTTPS（HTTP Secure ）
```
复制代码



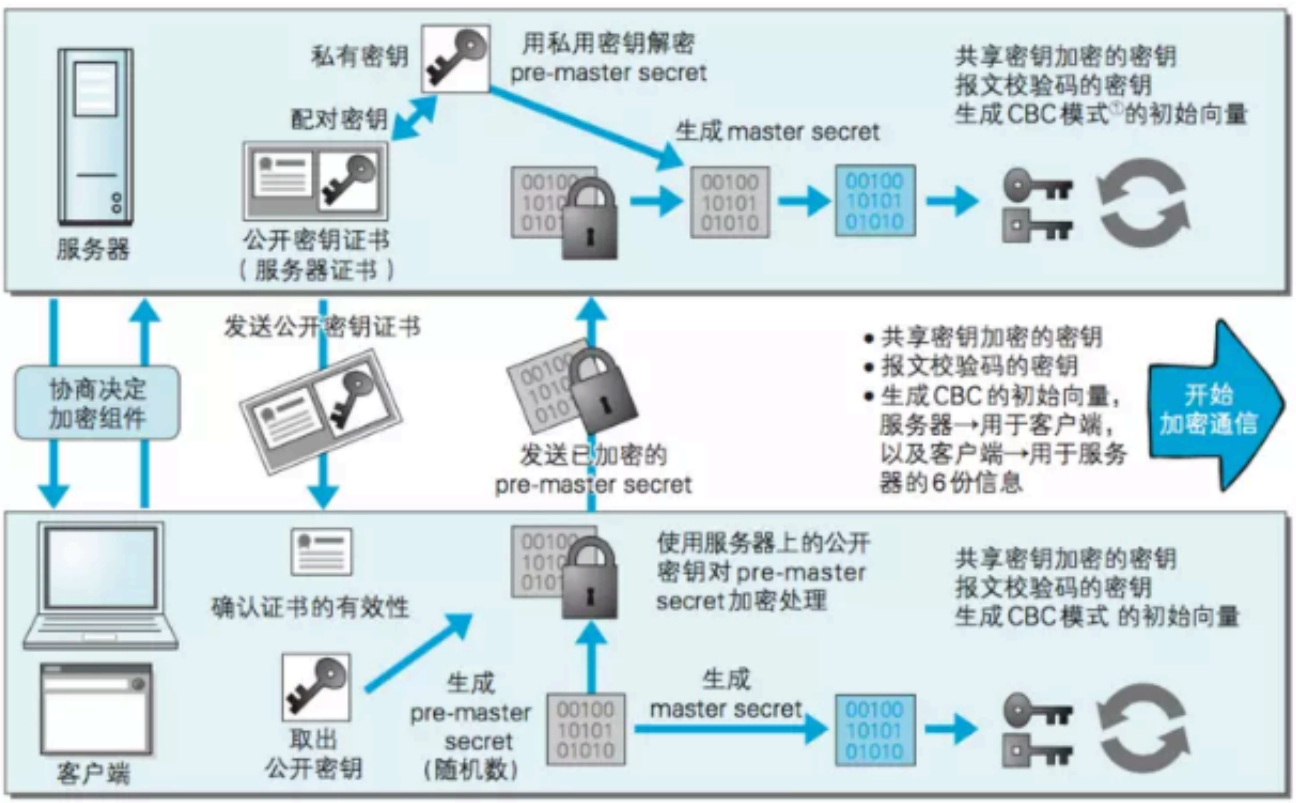
`HTTPS` 采用**共享密钥加密**（对称）和**公开密钥加密**（非对称）两者并用的**混合**加密机制。若密钥能够实现安全交换，那么有可能会考虑仅使用公开密钥加密来通信。但是公开密钥加密与共享密钥加密相比，其处理速度要慢。

> 所以应充分利用两者各自的优势，将多种方法组合起来用于通信。 在**交换密钥**阶段使用**公开密钥加密**方式，之后的建立通信**交换报文**阶段 则使用**共享密钥加密**方式。



图：混合加密机制

HTTPS 握手过程的简单描述如下：



1. 浏览器将自己支持的一套加密规则发送给网站。

```
服务器获得浏览器公钥
复制代码
```

2. 网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

```
浏览器获得服务器公钥
复制代码
```

3. 获得网站证书之后浏览器要做以下工作：

   (a). 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。

   (b). 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码（接下来通信的密钥），并用证书中提供的公钥加密（共享密钥加密）。

   (c) 使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

```
  浏览器验证 -> 随机密码
  服务器的公钥加密 -> 通信的密钥
  通信的密钥 -> 服务器
```

4. 网站接收浏览器发来的数据之后要做以下的操作：

   (a). 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。

   (b). 使用密码加密一段握手消息，发送给浏览器。

```
1   服务器用自己的私钥解出随机密码 -> 用密码解密握手消息（共享密钥通信）-> 验证HASH与浏览器是否一致（验证浏览器）
2
```

HTTPS的不足

- 加密解密过程复杂，导致访问速度慢
- 加密需要认向证机构付费
- 整个页面的请求都要使用HTTPS

面试题

在网页地址中输入了https://www.baidu.com,打开百度的首页,整个过程发生了什么?

1． 客户端连接到Web服务器

一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为403）建立一个TCP套接字连接。例如，https://www.baidu.com。

2．发送HTTP请求

通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。

3．服务器接受请求并返回HTTP响应

Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。

4．释放连接TCP连接

若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;

5 .客户端浏览器解析HTML内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

2、解析出 IP 地址后，根据该 IP 地址和默认端口 403，和服务器建立TCP连接

3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

5、释放 TCP连接;

6、浏览器将该 html 文本并显示内容;