01-HTTP (HyperText Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

Basically, HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

Building Blocks of the Web

There were three basic components devised by Tim Berners-Lee comprising the essence of Web technology:

1. A markup language for formatting hypertext documents.

2. A uniform notation scheme for addressing accessible resources over the network.

3. A protocol for transporting messages over the network.

The markup language that allowed cross-referencing of documents via hyperlinks was the HyperTextMarkupLanguage (HTML).

The uniform notation scheme is called the Uniform Resource Identifier (URI). it is most often referred to as the Uniform Resource Locator (URL).

HTTP is a core foundation of the World Wide Web. It was designed for transporting specialized messages over the network. Understanding of HTTP is just as critical in maintaining complex applications. Understanding the HTTP messages passed between servers, proxies and browsers leads to deeper insights into the nature of underlying problems.

The Uniform Resource Locator (URL)

Tim Berners-Lee knew that one piece of the Web puzzle would be a notation scheme for referencing accessible resources anywhere on the Internet. He devised this notational scheme

so that it would be flexible, so that it would be extensible, and so that it would support other protocols besides HTTP. This notational scheme is known as the URL or Uniform Resource Locator

Here is the generalized notation associated with URLs:

scheme://host[:port#]/path/.../[;url-params][?query-string][#anchor]

Let us break a URL down into its component parts:

البروتوكول الأساسي الذي سيتم استخدامه URL يعيّن هذا الجزء من عنوان

• *scheme*—this portion of the URL designates the underlying protocol to be used (e.g. 'http' or 'ftp'). This is the portion of the URL preceding the colon and two forward slashes.

• *host*—this is either the name of the IP address for the web server being accessed. This is usually the part of the URL immediately following the colon and two forward slashes.

• *port#*—this is an optional portion of the URL designating the port number that the target web server listens to. (The default port number for HTTP servers is 80, but some configurations are set up to use an alternate port number. When they do, that number must be specified in the URL.) The port number, if it appears, is found right after a colon that immediately follows the server name or address.

• *path*—logically speaking, this is the file system path from the 'root' directory of the server to the desired document. The path immediately follows the server and port number portions of the URL, and by definition includes that first forward slash.

• *url-params*—this one rarely used portion of the URL includes optional 'URL parameters'. It is now used somewhat more frequently, for session identifiers in web servers. If present, it follows a semi-colon immediately after the path information.

يتم استخدامه الآن

• *query-string*—this optional portion of the URL contains other dynamic parameters associated with the request. Usually, these parameters are produced as the result of user-entered variables in HTML forms. If present, the query string follows a question mark in the URL. Equal signs (=) separate the parameters from their values, and ampersands (&) mark the boundaries between parameter value pairs.

• *anchor*—this optional portion of the URL is a reference to a positional marker within the requested document, like a bookmark. If present, it follows a hash mark or pound sign ('#').

The breakout of a sample URL into components is illustrated below:

http://www.mywebsite.com/sj/test;id=8079?name=sviergn&x=true#stuff

SCHEME = http HOST = www.mywebsite.com PATH = /sj/test URL PARAMS = id=8079 QUERY STRING = name=sviergn&x=true ANCHOR = stuff

Note that the URL notation we are describing here applies to most protocols (e.g. http, https, and ftp).

HTTP Basic Features

There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.

- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

- **HTTP is stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

Multimedia Internet Mail Extensions (MIME)

قياسي ASCII في الأصل ، كانت أنظمة البريد الإلكتروني ترسل رسائل في شكل نص MIME

MIME Originally, e-mail systems transmitted messages in the form of standard ASCII text. If a user wanted to send a file in a non-text or 'binary' format (e.g. an image or sound file), it had to be encoded before it could be placed into the body of the message. The sender had to communicate the nature of the binary data directly to the receiver.

Multimedia Internet Mail Extensions (MIME) provided uniform mechanisms for including encoded attachments within a multipart e-mail message. MIME supports the definition of

boundaries separating the text portion of a message (the 'body') from its attachments, as well as the designation of attachment encoding methods.

MIME also supports the notion of content typing for attachments (and for the body of a message as well).

MIME-types are standard naming conventions for defining what type of data is contained in an attachment. A MIME-type is constructed as a combination of a top-level data type and a subtype. There is a fixed set of top-level data types, including 'text', 'image', 'audio', 'video', and 'application'. The subtypes describe the specific type of data, e.g. 'text/html', 'text/plain', 'image/jpeg', 'audio/mp3'.

The structure of HTTP messages

HTTP messages (both requests and responses) have a structure similar to e-mail messages; they consist of a block of lines comprising the message headers, followed by a blank line, followed by a message body. The structure of HTTP messages, however, is more sophisticated than the structure of e-mail messages.

Let us start with a very simple example: loading a static web page residing on a web server. A user may manually type a URL into her browser, she may click on a hyperlink found within the page she is viewing with the browser, or she may select a bookmarked page to visit. In each of these cases, the desire to visit a particular URL is translated by the browser into an HTTP request. An HTTP request message has the following structure:

METHOD /path-to-resource HTTP/version-number

Header-Name-1: value

Header-Name-2: value

[optional request body]

Every request starts with the special ***request line***, which contains a number of fields:

1. The 'method' represents one of several supported request methods, chief among them 'GET' and 'POST'.

2. The '/path-to-resource' represents the path portion of the requested URL.

3. The 'version-number' specifies the version of HTTP used by the client.

After the first line we see a list of HTTP headers, followed by a blank line, often called a <CR><LF> (for 'carriage return and line feed'). The blank line separates the request headers from the body of the request. The blank line is followed (optionally) by a body, which is in turn followed by another blank line indicating the end of the request message.

Here is a simplified version of the HTTP request message that would be transmitted to the web server at www.mywebsite.com:

GET /sj/index.html HTTP/1.1

Host: www.mywebsite.com

Note that the request message ends with a blank line. In the case of a GET request, there is no body, so the request simply ends with this blank line. Also, note the presence of a Host header. The server, upon receiving this request, attempts to generate a response message. An HTTP response message has the following structure:

HTTP/version-number status-code message

Header-Name-1: value

Header-Name-2: value

[response body]

The first line of an HTTP response message is the ***status line***. This line contains:

1. The version of HTTP being used.

2. Three-digit status code.
   شرح موجز مقروء بشري لرمز الحالة
3. Brief human-readable explanation of the status code.

This is a simplified version of the HTTP response message that the server would send back to the browser, assuming that the requested file exists and is accessible to the requestor:

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 9934 ...

<HTML>

<HEAD>

```
        <TITLE>SJs Web Page</TITLE>
    </HEAD>
    <BODY BGCOLOR="#ffffff">
        <H2 ALIGN="center">Welcome to Sviergn Jiernsen's Home Page</H2>
        ... </H2>
    </BODY>
</HTML>
```

Request methods

There are varieties of request methods specified in the HTTP protocol. The most basic ones defined in HTTP/1.1 are GET, HEAD, and POST. In addition, there are the less commonly used PUT, DELETE, TRACE, OPTIONS and CONNECT.

GET

The simplest of the request methods is GET. When you enter a URL in your browser, or click on a hyperlink to visit another page, the browser uses the GET method when making the request to the web server. GET requests date back to the very first versions of HTTP. A GET request does not have a body and, until the version 1.1, was not required to have headers. (*HTTP/1.1 requires that the Host header should be present in every request in order to support virtual hosting*)

The following example, we visited a URL, http://www.mywebsite.com/sj/index.html, using the GET method. Let's take a look at the request that gets submitted by an HTTP/1.1 browser when you fill out a simple HTML form to request a stock quote:

```
<HTML>
    <HEAD>
        <TITLE>Simple Form</TITLE>
    </HEAD>
    <BODY>
        <H2>Simple Form</H2>
            <FORM ACTION="http://finance.yahoo.com/q" METHOD="get"> Ticker:
```

<INPUT SIZE="25" NAME="s">

<INPUT TYPE="submit" VALUE="Get Quote">

</FORM>

<BODY>

</HTML>

If we enter 'YAHOO' in the form above, then the browser constructs a URL comprised of the 'ACTION' field from the form followed by a query string containing all of the form's input parameters and the values provided for them. The boundary separating the URL from the query string is a question mark. Thus, the URL constructed by the browser is http://www.finance.yahoo.com/q?s=YAHOO and the submitted request looks as follows:

GET /q?s=YAHOO HTTP/1.1

Host: finance.yahoo.com

User-Agent: Mozilla/4.75 [en] (WinNT; U)

The response that comes back from the server looks something like this:

HTTP/1.0 200 OK

Date: Sat, 03 Feb 2001 22:48:35 GMT

Connection: close

Content-Type: text/html

Set-Cookie: B=9ql5kgct7p2m3&b=2;expires=Thu,15 Apr 2010 20:00:00 GMT;

path=/; domain = . yahoo . com

<HTML>

<HEAD>

<TITLE>Yahoo! Finance - YHOO</TITLE>

</HEAD>

<BODY> ... </BODY>

</HTML>

POST

A fundamental difference between GET and POST requests is that POST requests have a body: content that follows the block of headers, with a blank line separating the headers from the body. Going back to the previous example, let's change the request method to POST and notice that the browser now puts form parameters into the body of the message, rather than appending parameters to the URL as part of a query string:

POST /q HTTP/1.1

Host: finance.yahoo.com

User-Agent: Mozilla/4.75 [en] (WinNT; U)

Content-Type: application/x-www-form-urlencoded

Content-Length: 6

s=YAHOO

Note that the URL constructed by the browser does not contain the form parameters in the query string. Instead, these parameters are included after the headers as part of the message body:

HTTP/1.0 200 OK

Date: Sat, 03 Feb 2001 22:48:35 GMT

Connection: close

Content-Type: text/html

Set-Cookie: B=9ql5kgct7p2m3&b=2;expires=Thu,15 Apr 2010 20:00:00 GMT; path=/; domain=.yahoo.com

<HTML>

<HEAD>

<TITLE>Yahoo! Finance - YHOO</TITLE>

</HEAD>

<BODY> ... </BODY>

</HTML>

Note that the response that arrives from finance.yahoo.com happens to be exactly the same as in the previous example using the GET method, but only because designers of the server application decided to support both request methods in the same way.

HEAD

Requests that use the HEAD method operate similarly to requests that use the GET method, except that the server sends back only headers in the response. This means the body of the request is not transmitted, and only the response metadata found in the headers is available to the client. This response metadata, however, may be sufficient to enable the client to make decisions about further processing, and may possibly reduce the overhead associated with requests that return the actual content in the message body.

ومع ذلك ، قد تكون البيانات

If we were to go back to the sample form and change the request method to HEAD, we would notice that the request does not change (except for replacing the word 'GET' with the word 'HEAD', of course), and the response contains the same headers as before but no body.

Status codes

The first line of a response is the status line, consisting of the protocol and its version number, followed by a three-digit status code and a brief explanation of that status code. The status code tells an HTTP client (browser or proxy) either that the response was generated as expected, or that the client needs to perform a specific action (that may be further parameterized via information in the headers). The explanation portion of the line is for human consumption; changing or omitting it will not cause a properly designed HTTP client to change its actions. Status codes are grouped into categories. HTTP Version 1.1 defines five categories of response messages:

تصنف على أنها إعلامية.
• 1xx—Status codes that start with '1' are classified as informational.

تشير إلى الاستجابات الناجحة.
• 2xx—Status codes that start with '2' indicate successful responses.

هي لأغراض إعادة التوجيه
• 3xx—Status codes that start with '3' are for purposes of redirection.

تمثل أخطاء طلب العميل
• 4xx—Status codes that start with '4' represent client request errors.

تمثل أخطاء الخادم
• 5xx—Status codes that start with '5' represent server errors.

Informational status codes (1xx)

These status codes serve solely informational purposes. They do not denote success or failure of a request, but rather impart information about how a request can be processed further. For example, a status code of “100” is used to tell the client that it may continue with a partially submitted request. Clients can specify a partially submitted request by including an ‘Expect’ header in the request message. A server can examine requests containing an ‘Expect’ header, determine whether or not it is capable of satisfying the request, and send an appropriate response. If the server is capable of satisfying the request, the response will contain a status code of ‘100’:

HTTP/1.1 100 Continue ...

If it cannot satisfy the request, it will send a response with a status code indicating a client request error, i.e. ‘417’:

HTTP/1.1 417 Expectation Failed ...

Successful response status codes (2xx)

مما يشير إلى أن الطلب قد اكتمل بنجاح وأن المورد المطلوب يتم إرساله مرة أخرى إلى العميل:

The most common successful response status code is ‘200’, which indicates that the request was successfully completed and that the requested resource is being sent back to the client:

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 9934 ...

```
<HTML>
    <HEAD>
        <TITLE>SJ’s Web Page</TITLE>
    </HEAD>
    <BODY BGCOLOR="#ffffff">
        <H2 ALIGN="center">Welcome to Sviergn Jiernsen’s Home Page</H2> ... </H2>
    </BODY>
</HTML>
```

Another example is ‘201’, which indicates that the request was satisfied and that a new resource was created on the server. مما يشير إلى تلبية الطلب وأنه تم إنشاء مورد جديد على الخادم

Redirection status codes (3xx)

الإشارة إلى أن الإجراءات الإضافية مطلوبة لتلبية الطلب الأصلي.

Status codes of the form ‘3xx’ indicate that additional actions are required to satisfy the original request. Normally this involves a redirection: the client is instructed to ‘redirect’ the request to another URL.

عادةً ما ينطوي هذا على إعادة توجيه: يُطلب من العميل "إعادة توجيه" الطلب إلى URL آخر.

كلاهما يوجه العميل للبحث عن المورد المطلوب أصلاً في الموقع الجديد المحدد في عنوان "الموقع" للاستجابة

For example, ‘301’ and ‘302’ both instruct the client to look for the originally requested resource at the new location specified in the ‘Location’ header of the response. The difference between the two is that ‘301’ tells the client that the resource has ‘Moved Permanently’, and that it should always look for that resource at the new location. ‘302’ tells the client that the resource has ‘Moved Temporarily’, and to consider this relocation a one-time deal, just for purposes of this request.

"يخبر العميل أن المورد "تم نقله بشكل دائم

في كلتا الحالتين ، يجب على العميل فور تلقي رد 301 أو 302 ، إنشاء وإرسال طلب جديد "تمت إعادة توجيهه" إلى الموقع الجديد. تحدث

In either case, the client should, immediately upon receiving a 301 or 302 response, construct and transmit a new request ‘redirected’ at the new location. Redirections happen all the time, often unbeknownst to the user. Browsers are designed to respond silently to redirection status codes, so that users never see redirection ‘happen’.

A perfect example of such silent redirection occurs when a user enters a URL specifying a directory, but leaving off the terminating slash. To visit Leon’s web site at Rutgers University, you could enter http://www.cs.rutgers. edu/~shklar in your browser. This would result in the following HTTP request:

GET /~shklar HTTP/1.1

Host: www.cs.rutgers.edu

But "~shklar" is actually a directory on the Rutgers web server, not a deliverable file. Web servers are designed to treat a URL ending in a slash as a request for a directory. Such requests may, depending on server configuration, return either a file with a default name (if present), e.g. index.html, or a listing of the directory’s contents. In either case, the web server must first redirect the request, from http://www.cs.rutgers.edu/~shklar to http://www.cs.rutgers.edu/~ shklar/, to properly present it:

HTTP/1.1 301 Moved Permanently

Location: http://www.cs.rutgers.edu/ ~shklar/

Content-Type: text/html ...

<html>

<head>

<title>301 Moved Permanently</title>

</head>

<body>

<h1>301 Moved Permanently</h1>

The document has moved here.

</body>

</html>

Today's sophisticated browsers are designed to react to '301' by updating an internal relocation table, so that in the future they can substitute the new address prior to submitting the request, and thus avoid the relocation response. To support older browsers that do not support automatic relocation, web servers still include a message body that explicitly includes a link to the new location. This affords the user an opportunity to manually jump to the new location.

Client request error status codes (4xx)

Status codes that start with '4' indicate a problem with the client request (e.g. '400 Bad Request'), an authorization challenge (e.g. '401 Not Authorized'), or the server's inability to find the requested resource (e.g. '404 Not Found').

Although '400', '401', and '404' are the most common in this category, some less common status codes are quite interesting. We have already seen (in the section on 'Informational Status Codes') an example of the use of '417 Expectation Failed'. In another example, the client might use the 'If-Unmodified-Since' header to request a resource only if it has not changed since a specific date:

GET/~shklar/HTTP/1.1

Host: www.cs.rutgers.edu

If-Unmodified-Since: Fri, 11 Feb 2000 22:28:00 GMT

Since this resource did change, the server sends back the '412 Precondition Failed' response:

HTTP/1.1 412 Precondition Failed

Date: Sun, 11 Feb 2001 22:28:31 GMT

Server: Apache/1.2.5

Server error status codes (5xx)

تشير إلى مشكلة في الخادم تمنعه من تلبية طلب صالح بخلاف ذلك

Finally, status codes that start with '5' indicate a server problem that prevents it from satisfying an otherwise valid request (e.g. '500 Internal Server Error' or '501 Not Implemented').

BETTER INFORMATION THROUGH HEADERS

As we already know, HTTP headers are a form of message metadata. Enlightened use of headers makes it possible to:

يتيح الاستخدام المستنير للرؤوس إمكانية:

إنشاء تطبيقات متطورة تنشئ الجلسات وتحافظ عليها

1. Construct sophisticated applications that establish and maintain sessions.

تعيين سياسات التخزين المؤقت والتحكم في المصادقة.

2. Set caching policies and control authentication.

تنفيذ منطق الأعمال.

3. Implement business logic.

The HTTP protocol specification makes a clear distinction between general headers, request headers, response headers, and entity headers. General headers apply to both request and response messages, but do not describe the body of the message. Examples of general headers include:

1• Date: Sun, 11 Feb 2001 22:28:31 GMT This header specifies the time and date that this message was created.

2 • Connection: Close This header indicates whether or not the client or server that generated the message intends to keep the connection open.

3 • Warning: Danger, Will Robinson! This header stores text for human consumption, something that would be useful when tracing a problem.

Request headers allow clients to pass additional information about themselves and about the request. For example:

1• User-Agent: Mozilla/4.75 [en] (WinNT; U) Identifies the software (e.g. a web browser) responsible for making the request.

2 • Host: www.neurozen.com This header was introduced to support virtual hosting, a feature that allows a web server to service more than one domain.

المرجع:

3. • Referer: http://www.cs.rutgers.edu/~shklar/index.html This header provides the server with context information about the request. If the request came about because a user clicked on a link found on a web page, this header contains the URL of that referring page.

تفويض

4. • Authorization: Basic [encoded-credentials] This header is transmitted with requests for resources that are restricted only to authorized users. Browsers will include this header after being notified of an authorization challenge via a response with a '401' status code. They consequently prompt users for their credentials (i.e. user id and password). They will continue to supply those credentials via this header in all further requests during the current browser session that access resources within the same authorization realm.

مساعدة الخادم على تمرير معلومات إضافية حول الاستجابة التي لا يمكن استنتاجها من رمز الحالة وحده.

Response headers help the server to pass additional information about the response that cannot be inferred from the status code alone. Here are some examples:

1. • Location: http://www.mywebsite.com/relocatedPage.html This header specifies a URL towards which the client should redirect its original request. It always accompanies the '301' and '302' status codes that direct clients to try a new location.

2. • WWW-Authenticate: Basic realm="KremlinFiles" This header accompanies the '401' status code that indicates an authorization challenge. The value in this header specifies the protected realm for which proper authorization credentials must be provided before the request can be processed. In the case of web browsers, the combination of the '401' status code and the WWW-Authenticate header causes users to be prompted for ids and passwords.

هذا العنوان غير مرتبط برمز حالة معين

3. • Server: Apache/1.2.5 This header is not tied to a particular status code. It is an optional header that identifies the server software.

Entity headers describe either message bodies or (in the case of request messages that have no body) target resources. Common entity headers include:

1. • Content-Type: mime-type/mime-subtype This header specifies the MIME type of the message body's content.

2. • Content-Length: xxx This optional header provides the length of the message body. Although it is optional, it is useful for clients such as web browsers that wish to impart information about the progress of a request. Where this header is omitted, the browser can only display the amount of data downloaded. But when the header is included, the browser can display the amount of data as a percentage of the total size of the message body.

إنه مفيد للعملاء

3• Last-Modified: Sun, 11 Feb 2001 22:28:31 GMT This header provides the last modification date of the content that is transmitted in the body of the message. It is critical for the proper functioning of caching mechanisms إنه أمر بالغ الأهمية للتشغيل السليم لآليات التخزين المؤقت

Type support through content-type

حتى الآن ، كنا نركز على البيانات الوصفية للرسالة ، ولسبب وجيه: فهم البيانات الوصفية أمر بالغ الأهمية لعملية إنشاء التطبيقات
So far, we were concentrating on message metadata, and for a good reason: understanding metadata is critical to the process of building applications. Still, somewhere along the line, there'd better be some content. After all, without content, Web applications would have nothing to present for end users to see and interact with. You've probably noticed that, when it comes to content you view on the Web, your browser might do one of several things. It might:

ربما لاحظت أنه عندما يتعلق الأمر بالمحتوى الذي تشاهده على الويب ، فقد يقوم المستعرض الخاص بك بأحد الأشياء العديدة. قد يكون:

- render the content as an HTML page,
- launch a helper application capable of presenting non-HTML content,
- present such content inline (within the browser window) through a plug-in, or
- get confused into showing the content of an HTML file as plain text without attempting to render it.

تشغيل تطبيق مساعد قادر على تقديم تقديم مثل •، HTML محتوى بخلاف هذا المحتوى مضمّنًا (داخل نافذة ، المتصفح) من خلال مكون إضافي أو• الخلط بين عرض محتوى ملف كنص عادي دون محاولة HTML عرضه.

من الواضح أن المتصفحات تفعل شيئًا ما لتحديد نوع المحتوى وتنفيذ الإجراءات المناسبة لهذا النوع. ماذا يجري هنا؟
What's going on here? Obviously, browsers do something to determine the content type and to perform actions appropriate for that type. HTTP borrows its content typing system from Multipurpose Internet Mail Extensions (MIME). MIME is the standard that was designed to help e-mail clients to display non-textual content.

محتوى غير نصي
As in MIME, the data type associated with the body of an HTTP message is defined via two-layer ordered encoding model, using Content-Type and Content-Encoding headers. In other words, for the body to be interpreted according to the type specified in the Content-Type header, it has to first be decoded according to the encoding method specified in the Content-Encoding header.

لكي يتم تفسير الجسم وفقًا للنوع المحدد في رأس ، نوع المحتوى

The Content-Encoding entity header is used to compress the media-type. When present, its value indicates which encodings were applied to the entity-body. It lets the client know, how to decode in order to obtain the media-type referenced by the Content-Type header.

عندما تكون ، موجودة فإنتشير القيمة

طرق ترميز المحتوى المحددة لرأس ترميز المحتوى هي ، HTTP / 1.1 في "deflate". و "compress" و "gzip"
In HTTP/1.1, defined content encoding methods for the Content-Encoding header are "gzip", "compress" and "deflate". The first two methods correspond to the formats produced by GNU

الطريقتان الأوليان تتطابقان مع التنسيقات التي أنتجها جنو

zip and UNIX compress programs. The third method, "deflate", corresponds to the zlib format associated with the deflate compression mechanism documented in RFC 1950 and 1951.

من الواضح ، إذا قامت خوادم الويب بتشفير المحتوى باستخدام طرق التشفير هذه ، فإن متصفحات الويب

Obviously, if web servers encode content using these encoding methods, web browsers (and other clients) must be able to perform the reverse operations on encoded message bodies prior to rendering or processing of the content.

Browsers are intelligent enough to open a compressed document file (e.g. test.doc.gz) and automatically invoke Microsoft Word to let you view the original test.doc file. It can do this if the web server includes the "Content-Encoding: gzip" header with the response. This header will cause a browser to decode the encoded content prior to presentation, revealing the test.doc document inside.

تلقائيًا للسماح لك Microsoft Word واستدعاء (test.doc.gz المتصفحات ذكية بما يكفي لفتح ملف مستند مضغوط (مثل مع "Content Encoding: gzip" الأصلي. يمكنه القيام بذلك إذا تضمن خادم الويب رأس test.doc بعرض ملف

The Content-Type header is set to a media-type that is defined as a combination of a type, subtype and any number of optional attribute/value pairs:

يتم تعيين رأس نوع المحتوى على نوع وسائط يتم تعريفه على أنه مزيج من النوع والنوع الفرعي وأي عدد من أزواج السمات / القيمة الاختيارية

media-type = type "/" subtype *(";" parameter-string) type = token subtype = token

The most common example is "Content-Type: text/html" where the type is set to "text" and the subtype is set to "html". This obviously tells a browser to render the message body as an HTML page.

Another example is:

Content-Type: text/plain; charset = 'us-ascii'

Here the subtype is "plain", plus there is a parameter string that is passed to whatever client program ends up processing the body whose content type is "text/plain". The parameter may have some impact on how the client program processes the content. If the parameter is not known to the program, it is simply ignored.

قد يكون للمعلمة بعض التأثير على كيفية معالجة برنامج العميل للمحتوى

Some other examples of MIME types are "text/xml" and "application/xml" for XML content, "application/pdf" for Adobe Portable Data Format, and "video/x-mpeg" for MPEG2 videos.