Lecture 02: Web Browsers

المسؤوليات الرئيسية للمتصفح

The main responsibilities of a browser are as follows:

إنشاء وإرسال الطلبات إلى خوادم الويب نيابة عن المستخدم

1. Generate and send requests to Web servers on the user's behalf, as a result of following hyperlinks, explicit typing of URLs, submitting forms, and parsing HTML pages that require auxiliary resources (e.g. images).

اقبل الاستجابات التي ترسلها خوادم الويب وقم بتفسيرها لإنتاج التمثيل المرئي الذي سيراه المستخدم.

2. Accept responses delivered by Web servers and interpret them to produce the visual representation to be viewed by the user. This will, at a bare minimum, involve examination of certain response headers such as Content-Type to determine what action needs to be taken and what sort of rendering is required.

اعرض النتائج في نافذة المتصفح أو من خلال أداة جهة خارجية ، اعتمادًا على نوع محتوى الاستجابة.

3. Render the results in the browser window or through a third party tool, depending on the content type of the response.

بناءً على رمز الحالة والعناوين في الاستجابة ، يتم استدعاء المتصفحات لأداء مهام أخرى ، بما في ذلك:

Depending on the status code and headers in the response, browsers are called upon to perform other tasks, including:

يجب أن يتخذ المستعرض قرارات بشأن ما إذا كان يحتاج إلى طلب البيانات من الخادم على الإطلاق أم لا.

1. *Caching*: the browser must make determinations as to whether or not it needs to request data from the server at all. It may have a cached copy of the same data item that it retrieved during a previous request. If so, and if this cached copy has not 'expired', the browser can eliminate a superfluous request for the resource. In other cases, the server can be queried to determine if the resource has been modified since it was originally retrieved and placed in the cache. Significant performance benefits can be achieved through caching.

قد يحتوي على

إذا كان الأمر كذلك ، وإذا لم "تنتهي صلاحية" هذه

2. *Authentication*: since web servers may require authorization credentials to access resources it has designated as secure, the browser must react to server requests for credentials, by prompting the user for authorization credentials, or by utilizing credentials it has already asked for in prior requests. نظرًا لأن خوادم الويب قد تتطلب بيانات اعتماد للوصول إلى الموارد التي عينتها على أنها آمنة ، يجب أن يتفاعل المتصفح مع

3. *State maintenance*: to record and maintain the state of a browser session across requests and responses, web servers may request that the browser accept cookies, which are sets of name/value pairs included in response headers. The browser must store the transmitted cookie information and make it available to be sent back in appropriate requests. In addition, the browser should provide configuration options to allow users the choice of accepting or rejecting cookies. صيانة الحالة: لتسجيل حالة جلسة المتصفح والحفاظ عليها عبر الطلبات والاستجابات ، قد تطلب خوادم الويب أن يقبل المتصفح ملفات تعريف الارتباط ، وهي مجموعات من أزواج الاسم / القيمة المضمنة في رؤوس الاستجابة. يجب أن يقوم المستعرض بتخزين معلومات ملفات تعريف

4. *Requesting supporting data items*: the typical web page contains images, sounds, and a variety of other ancillary objects. The proper rendering of the page is dependent upon the browser's retrieving those supporting data items for inclusion in the rendering process. This normally occurs transparently without user intervention. طلب عناصر البيانات الداعمة: تحتوي صفحة الويب النموذجية على صور وأصوات ومجموعة متنوعة من العناصر المساعدة الأخرى. يعتمد العرض

5. *Taking actions in response to other headers and status codes*: the HTTP headers and the status code do more than simply provide the data to be rendered by the browser. In some cases, they provide additional processing instructions, which may extend or supersede rendering information found elsewhere in the response. The presence of these instructions may

، ورمز الحالة أكثر من مجرد توفير البيانات التي سيقدمها المتصفح. في بعض الحالات HTTP اتخاذ إجراءات استجابة للرؤوس ورموز الحالة الأخرى: تعمل رؤوس
يقدمون تعليمات معالجة إضافية ، والتي قد توسع أو تحل محل معلومات العرض الموجودة في مكان آخر في الاستجابة.

قد يشير وجود هذه الإرشادات إلى وجود مشكلة في الوصول إلى المورد ، وقد يوجه المتصفح لإعادة توجيه ملف
طلب إلى مكان آخر. قد تشير أيضًا إلى أن الاتصال يجب أن يظل مفتوحًا ، لذلك
يمكن إرسال طلبات أخرى عبر نفس الاتصال. العديد من هذه الوظائف
المرتبطة بوظائف HTTP المتقدمة الموجودة في HTTP / 1.1

indicate a problem in accessing the resource, and may instruct the browser to redirect the request to another location. They may also indicate that the connection should be kept open, so that further requests can be sent over the same connection. Many of these functions are associated with advanced HTTP functionality found in HTTP/1.1. تدعم معظم متصفحات الويب أنواع المحتوى مثل text / html ، text / normal ، image / gif ، image /

6. *Rendering complex objects*: most web browsers inherently support content types such as text/html, text/plain, image/gif, and image/jpeg. This means that the browser provides native functionality to render objects with these contents inline: within the browser window, and without having to install additional software components. To render or play back other more complex objects (e.g. audio, video, and multimedia), a browser must provide support for these content types. Mechanisms must exist for invoking external helper applications or internal plug-ins that are required to display and playback these objects.

7. *Dealing with error conditions*: connection failures and invalid responses from servers are among the situations the browser must be equipped to deal with.

التعامل مع حالات الخطأ: تعد حالات فشل الاتصال والاستجابة غير الصالحة من الخوادم من بين المواقف
التي يجب أن يكون المتصفح مجهزًا للتعامل معها.

Architecture of Web Browser

The following list delineates the core functions associated with a Web browser. Each function can be thought of as a distinct module within the browser. Obviously these modules must communicate with each other in order to allow the browser to function, but they should each be designed atomically.

هذه الوحدة مسؤولة عن توفير الواجهة التي من خلالها يمكن للمستخدمين تتفاعل مع التطبيق. يتضمن ذلك تقديم وعرض
وتقديم النتيجة النهائية لمعالجة المتصفح للاستجابة المرسلة من الخادم.

• *User Interface*: this module is responsible for providing the interface through which users interact with the application. This includes presenting, displaying, and rendering the end result of the browser's processing of the response transmitted by the server.

هذه الوحدة تتحمل مسؤولية مهمة بناء طلبات HTTP لتقديمها إلى خوادم HTTP. عندما تُطلب من وحدة واجهة المستخدم

• *Request Generation*: this module bears responsibility for the task of building HTTP requests to be submitted to HTTP servers. When asked by the User Interface module or the Content Interpretation module to construct requests based on relative links, it must first resolve those links into absolute URLs.

• *Response Processing*: this module must parse the response, interpret it, and pass the result to the User Interface module. يجب على هذه الوحدة تحليل الاستجابة وتفسيرها وتمرير النتيجة إلى وحدة واجهة المستخدم.

هذه الوحدة مسؤولة عن اتصالات الشبكة

• *Networking*: this module is responsible for network communications. It takes requests passed to it by the Request Generation module and transmits them over the network to the appropriate Web server or proxy. It also accepts responses that arrive over the network and passes them to the Response Processing module. In the course of performing these tasks, it takes responsibility for establishing network connections and dealing with proxy servers specified in a user's network configuration options. يأخذ الطلبات التي يتم تمريرها إليه من خلال وحدة إنشاء الطلب وينقلها عبر الشبكة إلى خادم الويب أو
الوكيل المناسب. كما أنه يقبل الردود التى تصل عبر الشبكة ويمررها إلى وحدة معالجة الاستجابة. فى

بعد تلقي الاستجابة • *Content Interpretation*: having received the response, the Response Processing module needs help in parsing and deciphering the content. The content may be encoded, and this module is responding to decode it. Initial responses often have their content types set to text/html, but HTML responses embed or contain references to images, multimedia objects, JavaScript code, and style sheet information. This module performs the additional processing necessary for browser applications to understand these entities within a response. In addition, this module

يجب أن تخبر وحدة إنشاء الطلب بإنشاء طلبات إضافية لاسترداد المحتوى الإضافي مثل الصور والكائنات الأخرى.

must tell the Request Generation module to construct additional requests for the retrieval of auxiliary content such as images, and other objects.

يوفر التخزين المؤقت لمتصفحات الويب طريقة للاقتصاد من خلال تجنباسترداد غير ضروري للموارد التي يحتوي المتصفح بالفعل على نسخة قابلة للاستخدام

• *Caching*: caching provides web browsers with a way to economize by avoiding the unnecessary retrieval of resources that the browser already has a usable copy of, 'cached' away in local storage. Browsers can ask Web servers whether a desired resource has been modified since the time that the browser initially retrieved it and stored it in the cache. This module must provide facilities for storing copies of retrieved resources in the cache for later use, for accessing those copies when viable, and for managing the space (both memory and disk) allocated by the browser's configuration parameters for this purpose.

يمكن للمتصفحات أن تطلب من خوادم الويب ما إذا كان المورد المطلوب قد تمتم تعديله منذ

• *State Maintenance*: since HTTP is a stateless protocol, some mechanism must be in place to maintain the browser state between related requests and responses. Cookies are the mechanism of choice for performing this task, and support for cookies is in the responsibility of this module.

بروتوكول عديم الحالة ، يجب وضع آلية للحفاظ على حالة المتصفح بين الطلبات والاستجابات ذات الصلة. ملفات HTTP نظرًا لأن تعريف الارتباط هي الآلية المختارة لأداء هذه المهمة ، ويتحمل دعم ملفات تعريف الارتباط مسؤولية هذه الوحدة.

• *Authentication*: this module takes care of composing authorization credentials when requested by the server. It must interpret response headers demanding credentials by prompting the user to enter them (usually via a dialog). It must also store those credentials, but only for the duration of the current browser session, in case a request is made for another secured resource in what the server considers to be the same security 'realm'. (This absolves the user of the need to re-enter the credentials each time a request for such resources is made.)

تهتم هذه الوحدة بتكوين بيانات اعتماد التفويض

ترتيب

أخيرًا ، هناك عدد من خيارات التكوين التي يحتاج تطبيق المتصفح إلى دعمها.

• *Configuration*: finally, there are a number of configuration options that a browser application needs to support. Some of these are fixed, while others are user definable. This module maintains the fixed and variable configuration options for the browser, and provides an interface for users to modify those options under their control.

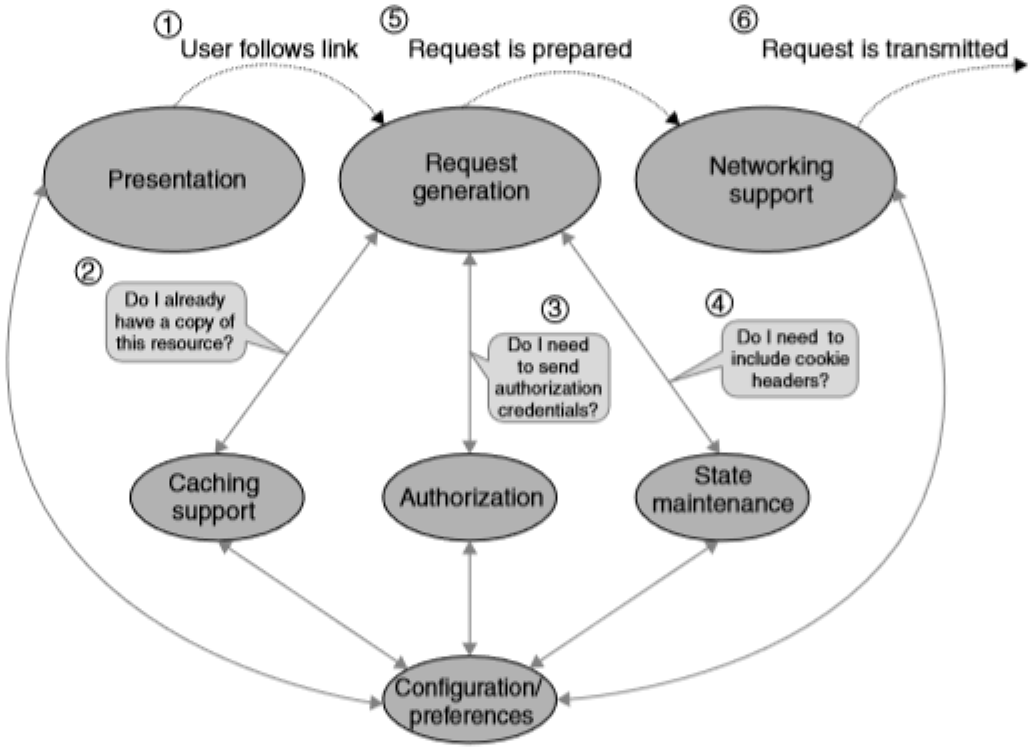
بينما يمكن تعريف البعض الآخر من قبل المستخدم

تحافظ هذه الوحدة على خيارات التكوين الثابتة والمتغيرة للمتصفح ، وتوفر واجهة للمستخدمين لتعديل تلك الخيارات تحت سيطرتهم.

PROCESSING FLOW

The following diagram shows the processing flow for the creation and transmission of a request in a typical browser. We begin with a link followed by a user. Users can click on hyperlinks presented in the browser display window, they might choose links from lists of previously visited links (history or bookmarks), or they might enter a URL manually. In each of these cases, processing begins with the User Interface module, which is responsible for presenting the display window and giving users access to browser functions (e.g. through menus and shortcut keys).

In general, an application using a GUI (graphical user interface) operates using an event model. User actions—clicking on highlighted hyperlinks, for example—are considered events that must be interpreted properly by the User Interface module



• *Entering URLs manually*: usually, this is accomplished by providing a text entry box in which the user can enter a URL, as well as through a menu option (File→ Open) that opens a dialog box for similar manual entry.

اختيار الروابط التي تمت زيارتها مسبقًا:

• *Selecting previously visited links*: the existence of this mechanism, naturally, implies that the User Interface module must also provide a mechanism for maintaining a history of visited links. The maximum amount of time that such links will be maintained in this list, as well as the maximum size to which this list can grow, can be established as a user-definable parameter in the Configuration module. The 'Location' or 'Address' text area in the browser window can be a dropdown field that allows the user to select from recently visited links. The 'Back' button allows users to go back to the page they were visiting previously. In addition, users should be able to save particular links as "bookmarks", and then access these links through the user interface at a later date.

تحديد الارتباطات التشعبية المعروضة:

• *Selecting displayed hyperlinks*: there are a number of ways for users to select links displayed on the presented page. In desktop browsers, the mouse click is probably the most common mechanism for users to select a displayed link, but there are other mechanisms on the desktop and on other platforms as well. Since the User Interface module is already responsible for rendering text according to the specifications found in the page's HTML markup, it is also responsible for doing some sort of formatting to highlight a link so that it stands out from other text on the page. Most desktop browsers also change the cursor shape when the mouse is ‘over’ a hyperlink, indicating that this is a valid place for users to click. Highlighting mechanisms vary for non-desktop platforms, but they should always be present in some form.

Once the selected or entered link is passed on to the Request Generation module, it must be resolved. Links found on a displayed page can be either absolute or relative. Absolute URLs are complete URLs, containing all the required URL components, e.g. protocol://host/path. These do not need to be resolved and can be processed without further intervention. A relative URL specifies a location relative to:

يجب حلها

1. the current location being displayed (i.e. the entire URL including the path, up to the directory in which the current URL resides), when the HREF contains a relative path that does not begin with a slash, e.g.: <A HREF="some directory/ just a file name.html">), or

2. the current location’s web server root (i.e., only the host portion of the URL), when the HREF contains a relative path that does begin with a slash, e.g. <A HREF="/root level directory/another file name.html">.

Current URL: http://www.myserver.com/mydirectory/index.html

<A HREF ="anotherdirectory/page2.html">...</A> →
http://www.myserver.com/mydirectory/anotherdirectory/page2.html

<A HREF ="/rootleveldirectory/homepage.html">...</A>
http://www.myserver.com/rootleveldirectory/homepage.html

Current URL: http://www.myserver.com/mydirectory/anotherpage.html

<A HREF ="anotherdirectory/page2.html">...</A>
http://www.myserver.com/mydirectory/anotherdirectory/page2.html

<A HREF ="/yetanotherdirectory/homepage.html">...</A>
http://www.myserver.com/yetanotherdirectory/homepage.html

Current URL: http://www.myserver.com/mydirectory/differentpage.html

<BASE HREF ="http://www.yourserver.com/otherdir/something.html">

<A HREF ="anotherdirectory/page2.html">...</A>
http://www.yourserver.com/otherdir/anotherdirectory/page2.html

<A HREF ="/yetanotherdirectory/homepage.html">...</A>
http://www.yourserver.com/yetanotherdirectory/homepage.html

The process of resolution changes if an optional <BASE HREF ="..."> tag is found in the HEAD section of the page. The URL specified in this tag replaces the current location as the “base” from which resolution occurs in the previous examples. Once the URL has been resolved, the Request Generation module builds the request, which is ultimately passed to the Networking

لإنجاز هذه المهمة ، يجب أن تقوم وحدة إنشاء الطلب
التواصل مع مكونات المتصفح الأخرى:

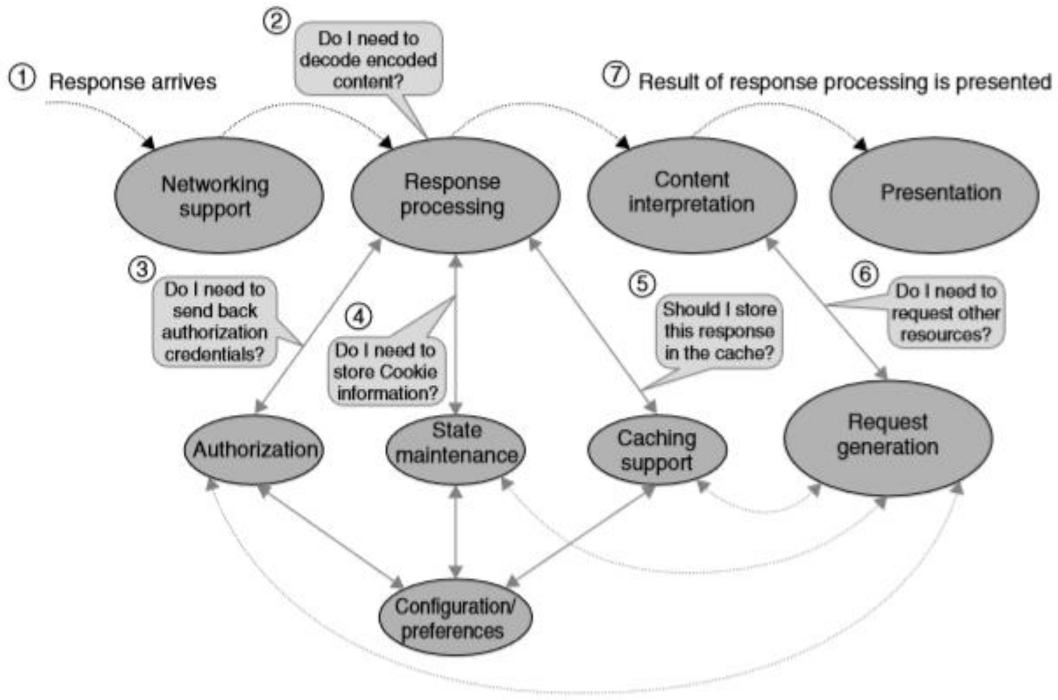
module for transmission. To accomplish this task, the Request Generation module has to communicate with other browser components:

• It asks the Caching module "Do I already have a copy of this resource?" If so, it needs to determine whether it can simply use this copy, or whether it needs to ask the server if the resource has been modified since the browser cached a copy of this resource.

• It asks the Authorization module "Do I need to include authentication credentials in this request?" If the browser has not already stored credentials for the appropriate domain, it may need to contact the User Interface module, which prompts the user for credentials.

• It asks the State Mechanism module "Do I need to include Cookie headers in this request?" It must determine whether the requested URL matches domain and path patterns associated with previously stored cookies.

The constructed request is passed to the Networking module so it can be transmitted over the network.


① Response arrives
② Do I need to decode encoded content?
⑦ Result of response processing is presented
Networking support
Response processing
Content interpretation
Presentation
③ Do I need to send back authorization credentials?
④ Do I need to store Cookie information?
⑤ Should I store this response in the cache?
⑥ Do I need to request other resources?
Authorization
State maintenance
Caching support
Request generation
Configuration/ preferences

Once a request has been transmitted, the browser waits to receive a response. It may submit additional requests while waiting. Requests may have to be resubmitted if the connection is closed before the corresponding responses are received. It is the server's responsibility to transmit responses in the same order as the corresponding requests were received. However, the browser is responsible for dealing with servers that do not properly maintain this order, by delaying the processing of responses that arrive out of sequence. The above diagram describes the flow for this process. A response is received by the Networking module, which passes it to the Response Processing module. This module must also cooperate and communicate with other modules to do its job. It examines response headers to determine required actions.

• If the status code of the response is 401 Not Authorized, this means that the request lacked necessary authorization credentials. The Response Processing module asks the Authorization module whether any existing credentials might be used to satisfy the request. The Authorization module may, in turn, contact the User Interface module, which would prompt the user to enter authorization credentials. In either case, this results in the original request being retransmitted with an Authorization header containing the required credentials.

• If the response contains Set-Cookie headers, the State Maintenance module must store the cookie information using the browser's persistence mechanism.

Next, the response is passed to the Content Interpretation module, which has a number of responsibilities:

• If the response contains Content-Transfer-Encoding and/or Content Encoding headers, the module needs to decode the body of the response.

• The module examines the Cache-Control, Expires, to determine whether the browser needs to cache the decoded content of the response. If so, the Caching module is contacted to create a new cache entry or update an existing one.

• The Content-Type header determines the MIME type of the response content. Different MIME types, naturally, require different kinds of content processing. Modern browsers support a variety of content types natively, including HTML (text/html), graphical images (image/gif, and image/jpeg), and sounds (audio/wav). Native support means that processing of these content types is performed by built-in browser components. Thus, the Content Interpretation module must provide robust support for such processing. Leading edge browsers already provide support for additional content types, including vector graphics and XSL style sheets.

• For MIME types that are not processed natively, browsers usually provide support mechanisms for the association of MIME types with helper applications and plug-ins. Helper applications render content by invoking an external program that executes independent of the browser, while plug-ins render content within the browser window. The Content Interpretation module must communicate with the Configuration module to determine what plug-ins are installed and what helper application associations have been established, to take appropriate action when receiving content that is not natively supported by the browser. This involves a degree of interaction with the operating system, to determine system-level associations configured for filename extensions, MIME types, and application programs. However, many browsers override (or even completely ignore) these settings, managing their own sets of associations through the Configuration module.

• Some content types (e.g. markup languages, Flash movies) may embed references to other resources needed to satisfy the request. For instance, HTML pages may include references to images or JavaScript components. The Content Interpretation module must parse the content prior to passing it on to the User Interface module, determining if additional requests will be needed. If so, URLs associated with these requests get resolved when they are passed to the Request Generation module.

As each of the requested resources arrives in sequence, it is passed to the User Interface module so that it may be incorporated in the final presentation. The Networking module maintains its queue of requests and responses, ensuring that all requests have been satisfied, and resubmitting any outstanding requests.

All along the way, various subordinate modules are asked questions to determine the course of processing (including whether or not particular tasks need to be performed at all). For example, the Content Interpretation module may say ‘This page has IMG tags, so we must send HTTP requests to retrieve the associated images,’ but the Caching module may respond by saying ‘We already have a usable copy of that resource, so don’t bother sending a request to the network for it.’ (Alternatively, it may say ‘We have a copy of that resource, but let’s ask the server if its copy of the resource is more recent; if it’s not, it doesn’t need to send it back to us.’) Or the Configuration module may say ‘No, don’t send a request for the images on this page, this user has a slow connection and has elected not to see images.’ Or the State Maintenance mechanism may jump in and say ‘Wait, we’ve been to this site before, so send along this identifying cookie information with our requests.’

PROCESSING HTTP REQUESTS AND RESPONSES

Let us examine how browsers build and transmit HTTP requests, and how they receive, interpret, and present HTTP responses. After we have covered the basics of constructing requests and interpreting responses, we can look at the more complex interactions involved when HTTP transactions involve caching, authorization, cookies, request of supporting data items, and multimedia support.

HTTP requests

يتكون من خطوتين أساسيتين: إنشاء طلب HTTP ، وإنشاء اتصال لنقله عبر الإنترنت إلى الخادم الهدف أو وكيل وسيط.

The act of sending an HTTP request to a web server, in its most trivial form, consists of two basic steps: constructing the HTTP request, and establishing a connection to transmit it across the Internet to the target server or an intermediate proxy. The construction of requests is the responsibility of the Request Generation module. Once a request has been properly constructed, this module passes it to the Networking module, which opens the socket to transmit it either directly to the server or to a proxy. Before the Request Generation module has even begun the process of building the request, it needs to ask a whole series of questions of the other modules:

1. Do I already have a cached copy of this resource? If an entry exists in the cache that satisfies this same request, then the transmitted request should include an If-Modified-Since header, containing the last modification time associated with the stored cache entry. If the resource found on the server has not been modified since that time, the response will come back with a 304 Not Modified status code, and that cache entry can be passed directly to the User Interface module.

2. Is there any additional information I need to send as part of this request? If this request is part of a series of requests made to a particular web server, or if the target web server has been visited previously, it may have sent “state” information (in the form of Set-Cookie headers) to the browser. The browser must set and maintain cookies according to the server instructions: either for a specified period of time or for the duration of the current session. In addition, the set of saved cookies must be examined prior to sending a request to determine whether cookie information needs to be included in that request. (State Maintenance)

3. Is there any other additional information I need to send as part of this request? If this resource is part of an authorization realm for which the user has already supplied authentication credentials, those credentials should be stored by the browser for the duration of a session, and should be supplied with requests for other resources in the same realm. (Authorization)

User preferences may modify the nature of the request, possibly even eliminating the need for one entirely. For example, users may set a preference via the Configuration module telling the browser not to request images found within an HTML page. They can turn off Java applet support, meaning that requests for applets need not be processed. They can also instruct the browser to reject cookies, meaning that the browser does not need to worry about including Cookie headers in generated requests.

In the previous lecture we studied the HTTP protocol, we described the general structure of HTTP requests, and provided some examples. To refresh our memories, here is the format of an HTTP request:

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

Header-Name-1: value

Header-Name-2: value

[optional request body]

An HTTP request contains a request line, followed by a series of headers (one per line), followed by a blank line. The blank line may serve as a separator, delimiting the headers from an optional body portion of the request. A typical example of an HTTP request might look something like this:

POST /update.cgi HTTP/1.0

Host: www.somewhere.com

Referer: http://www.somewhere.com/formentry.html

name=joe&type=info&amount=5

The process of constructing an HTTP request typically begins when a web site visitor sees a link on a page and clicks on it, telling the browser to present the content associated with that link. There are other possibilities, such as entering a URL manually, or a browser connecting to a default home page when starting up, but this example allows us to describe typical browser activity more comprehensively.

Constructing the request line When a link is selected, the browser's User Interface module reacts to an event. A GUI-based application operates using an event model, in which user actions (e.g. typing, mouse clicking) are translated into events that the application responds to. In response to a mouse click on a hyperlink, for example, the User Interface module determines and resolves the URL associated with that link, and passes it to the Request Generation module.

At this point, the Request Generation module begins to construct the request. The first portion of the request that needs to be created is the request line, which contains a 'method' (representing one of several supported request methods), the '/path-to-resource' (representing the path portion of the requested URL), and the 'version-number' (specifying the version of HTTP associated with the request).

Let's examine these in reverse order:

The 'version-number' should be either HTTP/1.1 or HTTP/1.0. A modern up-to-date client program should always seek to use the latest version of its chosen transmission protocol, unless the recipient of the request is not sophisticated enough to make use of that latest version. Thus, at the present time, a browser should seek to communicate with a server using HTTP/1.1, and should only 'fall back' to HTTP/1.0 if the server with which it is communicating does not support HTTP/1.1.

The 'path-to-resource' portion is a little more complicated, and is in fact dependent on which version of HTTP is employed in the request. You may remember that this portion of the request line is supposed to contain the "path" portion of the URL. This is the part of the URL following the host portion of the URL (i.e. "http://hostname"), starting with the "/".

مهممممم

HTTP / 1.0 يتطلب تضمين عنوان URL بالكامل الطلبات

The situation is complicated when the browser connects to a proxy server to send a request, rather than connecting directly to the target server. Proxies need to know where to forward the request. If only the path-to-resource portion is included in the request line, a proxy would have no way of knowing the intended destination of the request. HTTP/1.0 requires the inclusion of the entire URL for requests directed at proxy servers, but forbids the inclusion of the entire URL for requests that get sent directly to their target servers. This is because HTTP/1.0 servers do not understand requests where the full URL is specified in the request line. لا تفهم الطلبات حيث HTTP / 1.0 وذلك لأن خوادم الكامل في سطر الطلب يتم تحديد عنوان URL.

في المقابل ، يتوقع وكلاء HTTP / 1.0 أن تحتوي الطلبات الواردة على عناوين URL كاملة.

In contrast, HTTP/1.0 proxies expect that incoming requests contain full URLs. When HTTP/1.0 proxies reconstruct requests to be sent directly to their target servers, they remove the server portion of the request URL. When requests must pass through additional proxies, this reconstruction is not performed, and the requests remain unchanged.

HTTP / 1.1 أكثر مرونة يجعل إدراج عنوان URL بالكامل في سطر الطلب مقبولاً في جميع المواقف ، بغض النظر عما إذا كان الوكيل متضمنًا أم لا

تتطلب أن تتضمن

HTTP/1.1 is more flexible; it makes the inclusion of the entire URL on the request line acceptable in all situations, irrespective of whether a proxy is involved. However, to facilitate this flexibility, HTTP/1.1 requires that submitted requests all include a "Host" header, specifying the IP address or name of the target server. This header was originally introduced to support virtual hosting, a feature that allows a web server to service more than one domain. This means that a single Web server program could be running on a server machine, accepting requests associated with many different domains. However, this header also provides sufficient information to proxies so that they can properly forward requests to other servers/proxies. Unlike HTTP/1.0 proxies, HTTP/1.1 proxies do not need to perform any transformation of these requests. إلى إجراء أي تحويل لهذه الطلبات HTTP / 1.1 لا تحتاج وكلاء ، HTTP / 1.0 بخلاف بروكسيات

The ‘method’ portion of the request line is dependent on which request method is specified, implicitly or explicitly. When a hyperlink (textual or image) is selected and clicked, the GET method is implicitly selected. In the case of HTML forms, a particular request method may be specified in the <FORM> tag:

<FORM ACTION ="http://www.somewhere.com/update.cgi" METHOD ="POST"> ... </FORM>

As mentioned in the chapter on the HTTP protocol, the GET method represents the simplest format for HTTP requests: a request line, followed by headers, and no body. Other request methods such as POST and PUT make use of a request body that follows the request line, headers, and blank line. (The blank line serves as a separator delimiting the headers from the body.) This request body may contain parameters associated with an HTML form, a file to be uploaded, or a combination of both. In any case, we are still working on the construction of the request line.

The ‘method’ portion will be set to "GET" by default: for textual or image-based hyperlinks that are followed, and for forms that do not explicitly specify a METHOD. If a form does explicitly specify a METHOD, that method will be used instead.

Constructing the headers Next, we come to the headers. There are a number of headers that a browser should include in the request:

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. This means that a single Web server program could be running on a server machine, accepting requests associated with many different domains. Without this header, the Web server program could not tell which of its many domains the target of the request was. In addition, this header provides information to proxies to facilitate proper routing of requests.

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

Identifies the software (e.g. a web browser) responsible for making the request. Your browser (or for that matter any Web client) should provide this information to identify itself to servers. The conventions are to produce a header containing the name of the product, the version number, the language this particular copy of the software uses, and the platform it runs on: Product/version. number [lang] (Platform)

Referer: http://www.cs.rutgers.edu/~shklar/index.html إذا تم إنشاء هذا الطلب لأن المستخدم حدد ارتباطًا موجودًا على صفحة ويب ، فيجب أن يحتوي هذا العنوان على عنوان URL لصفحة الإحالة.

If this request was instantiated because a user selected a link found on a web page, this header should contain the URL of that referring page. Your Web client should keep track of the current URL it is displaying, and it should be sure to include that URL in a Referer header whenever a link on the current page is selected.

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

This header specifies the time and date that this message was created. All request and response messages should include this header.

Accept: text/html, text/plain ,type/subtype, ...

Accept- Charset: ISO-8859-1,character −set −identifier, ...

Accept-Language: en, language −identifier, ...

Accept-Encoding: compress, gzip,...

These headers list the MIME types, character sets, languages, and encoding schemes that your client will 'accept' in a response from the server. If your client needs to limit responses to a finite set, then these should be included in these headers. Your client's preferences with respect to these items can be ranked by adding relative values in the form of q=qvalue parameters, where qvalue is a digit.

Content-Type:*mime-type/mime-subtype*

Content-Length: xxx

These entity headers provide information about the message body. For POST and PUT requests, the server needs to know the MIME type of the content found in the body of the request, as well as the length of the body.

Cookie: name=value

This request header contains cookie information that the browser has found in responses previously received from Web servers. This information needs to be sent back to those same servers in subsequent requests, maintaining the 'state' of a browser session by providing a name-value combination that uniquely identifies a particular user. Interaction with the State Maintenance module will determine whether these headers need to be included in requests, and if so what their values should be. Note that a request will contain multiple Cookie headers if there is more than one cookie that should be included in the request.

Authorization: SCHEME encoded-user id: password

This request header provides authorization credentials to the server in response to an authentication challenge received in an earlier response. The scheme (usually 'basic') is followed by a string composed of the user ID and password (separated by a colon), encoded in the base64 format. Interaction with the Authorization module will determine what the content of this header should be.

Constructing the request body This step of the request construction process applies only for methods like POST and PUT that attach a message body to a request. The simplest example is that of including form parameters in the message body when using the POST method. They must be URL-encoded to enable proper parsing by the server, and thus the Content-Type header in the request must be set to application/x-www-form-urlencoded. There are more complex uses for the request body. File uploads can be performed through forms employing the POST method (using multipart MIME types), or (with the proper server security configuration) web resources can be modified or created directly using the PUT method. With the PUT method, the ContentType of the request should be set to the MIME type of the content that is being uploaded. With the POST method, the Content-Type of the request should be set to multipart/form-data, while the Content-Type of the individual parts should be set to the MIME type of those parts. This Content-Type header requires the "boundary" parameter, which specifies a string of text that separates discrete pieces of content found in the body:

... Content-Type: multipart/ multipart subtype ; boundary=" random-string "

-random-string

Content-Type: type/subtype of part 1

Content-Transfer-Encoding: encoding scheme for part 1

content of part 1

-random-string

Content-Type: type/subtype of part 2

Content-Transfer-Encoding: encoding scheme for part 2

content of part 2

Note that each part specifies its own Content-Type, and its own Content-Transfer-Encoding. This means that one part can be textual, with no encoding specified, while another part can be binary (e.g. an image), encoded in Base64 format, as in the following example:

... Content-Type: multipart/form-data; boundary="gc0p4Jq0M2Yt08jU534c0p"

--gc0p4Jq0M2Yt08jU534c0p

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

&filename= ... &param =value

--gc0p4Jq0M2Yt08jU534c0p

Content-Type: image/gif

Content-Transfer-Encoding: base64

FsZCBoYWQgYSBmYXJtCkUgSST2xkIE1hY0Rvbm GlzIGZhcm0gaGUgaGFkBFIEkgTwpBbmQgb24ga IHKRSBJIEUgSSBPCldpdGggYSNvbWUgZHVja3M BxdWjayBoZXJlLApFjayBxdWFhIHF1YWNrIHF1 XJlLApldmVyeSB3aGYWNrIHRoZVyZSBhIHF1YW NrIHF1YWNrCEkgTwokUgSSBFI=

Transmission of the request Once the request has been fully constructed, it is passed to the Networking module, which transmits the request.

This module must first determine the target of the request. Normally, this can be obtained by parsing the URL associated with the request. However, if the browser is configured to employ a proxy server, the target of the request would be that proxy server. Thus, the Configuration module must be queried to determine the actual target for the request. Once this is done, a socket is opened to the appropriate machine.

HTTP responses

In the request/response paradigm, the transmission of a request anticipates the receipt of some sort of a response. Hence, browsers and other Web clients must be prepared to process HTTP responses. This task is the responsibility of the Response Processing module. As we know, HTTP responses have the following format:

HTTP/version-number status-code message

Header-Name-1: value Header-Name-2: value

[response body]

An HTTP response message consists of a status line (containing the HTTP version, a three-digit status code, and a brief human-readable explanation of the status code), a series of headers (again, one per line), a blank line, and finally the body of the response. The following is an example of the HTTP response message that a server would send back to the browser when it is able to satisfy the incoming request:

```
HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 1234 ...

<HTML>

    <HEAD> <TITLE>...</TITLE>

    </HEAD>

    <BODY BGCOLOR="#ffffff">

        <H2 ALIGN="center">...</H2> ... </H2>

    </BODY>

</HTML>
```

In this case, we have a successful response: the server was able to satisfy the client’s request and sent back the requested data. Now, of course, the requesting client must know what to do with this data. When the Networking module receives a response, it passes it to the Response Processing module.

First, this module must interpret the status code and header information found in the response to determine what action it should take. It begins by examining the status code found in the first line of the response (the status line). In the lecture covering the HTTP protocol, we delineated the different classes of status codes that might be sent by a Web server:

• informational status codes (1xx),

• successful response status codes (2xx),

• redirection status codes (3xx), • client request error status codes (4xx),

and • server error status codes (5xx).

Obviously, different actions need to be taken depending on which status code is contained in the response. Since the successful response represents the simplest and most common case, we will begin with the status code "200".

Processing successful responses The status code "200" represents a successful response, as indicated by its associated message "OK". This status code indicates that the browser or client should take the associated content and render it in accordance with the specifications included in the headers:

Content-Transfer-Encoding: chunked

Content-Encoding: compress | *gzip*

The presence of these headers indicates that the response content has been encoded and that, prior to doing anything with this content, it must be de-coded.

Content-Type:mime-type/mime-subtype

This header specifies the MIME type of the message body's content. Browsers are likely to have individualized rendering modules for different MIME types. For example, text/html would cause the HTML rendering module to be invoked, text/plain would make use of the plain text rendering module, and image/gif would employ the image rendering module. Browsers provide built-in support for a limited number of MIME types, while deferring processing of other MIME types to plug-ins and helper applications.

Content-Length: xxx

This optional header provides the length of the message body in bytes. Although it is optional, when it is provided a client may use it to impart information about the progress of a request. When the header is included, the browser can display not only the amount of data downloaded, but it can also display that amount as a percentage of the total size of the message body.

Set-Cookie: name=value ; domain= domain.name ; path= path-within-server ; [secure]

If the server wishes to establish a persistent mechanism for maintaining session state with the user's browser, it includes this header along with identifying information. The browser is responsible for sending back this information in any requests it makes for resources within the same domain and path, using Cookie headers. The State Maintenance module stores cookie information found in the response's Set-Cookie headers, so that the browser can later retrieve that information for Cookie headers it needs to include in generated requests. Note that a response can contain multiple Set-Cookie headers.

Cache-Control: private | no-cache |... Pragma: no-cache

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

These headers influence caching behavior. Depending on their presence or absence (and on the values they contain), the Caching Support module will decide whether the content should be cached, and if so, for how long (e.g. for a specified period of time or only for the duration of this browser session).

Once the content of a successful response has been decoded and cached, the cookie information contained in the response has been stored, and the content type has been determined, then the response content is passed on to the Content Interpretation module.

This module delegates processing to an appropriate sub-module, based on the content type. For instance, images (Content-Type: image/*) are processed by code devoted to rendering images. HTML content(Content-Type: text/html) is passed to HTML rendering functions, which would in turn pass off processing to other functions. For instance, JavaScript—contained within <SCRIPT> block tags or requested via references to URLs in <SCRIPTSRC=...>tags—must be interpreted and processed appropriately. In addition, style sheet information embedded in the page must also be processed.

Only after all of this processing is complete is the resulting page passed to the User Interface module to be displayed in the browser window.

There are other status codes that fit into the ‘successful response’ category (2xx) including: "201 Created": a new resource was created in response to the request, and the Location header contains the URL of the new resource. "202 Accepted": the request was accepted, but may or may not be processed by the server.

"204 No Content": nobody was included with the response, so there is no content to present. This tells the browser not to refresh or update its current presentation as a result of processing this request.

"205 Reset Content": this is usually a response to a form processed for data entry. It indicates that the server has processed the request, and that the browser should retain the current presentation, but that it should clear all form fields. Although these status codes are used less often than the popular 200 OK, browsers should be capable of interpreting and processing them appropriately.

Processing of responses with other status codes Aside from the successful status code of 200, the most common status codes are the ones associated with redirection (3xx) and client request errors (4xx).

Client request errors are usually relatively simple to process: either the browser has somehow provided an invalid request (400 Bad Request), or the URL the browser requested couldn’t be found on the server (404 Not Found). In either of these cases, the browser simply presents a message indicating this state of affairs to the user.

Authentication challenges that are caused by the browser attempting to access protected resources (e.g. 401 Not Authorized) are also classified as ‘client error’ conditions. Some Web servers may be configured to provide custom HTML presentations when one of these conditions occurs. In those situations, the browser should simply render the HTML page included in the response body:

```
HTTP/1.1 404 Not Found

Content-Type: text/html

<HTML>

        <HEAD> <TITLE>Whoops!</TITLE> </HEAD>

        <BODY BGCOLOR="#ffffff">

                <h3>Look What You’ve Done!</h3> You’ve broken the internet! <P> (Just kidding, you
                simply requested an invalid address on this site.)

        </BODY>

</HTML>
```

Redirection status codes are also relatively simple to process. They come in two varieties: 301 Moved Permanently and 302 Moved Temporarily. The processing for each of these is similar. For responses associated with each of these status codes, there will be a Location: header present. The browser needs to submit a further request to the URL specified in this header to perform the desired redirection. Some Web servers may be configured to include custom HTML bodies when one of these conditions arises. This is for the benefit of older browsers that do not support automatic redirection and default to rendering the body when they don’t recognize the status code. Browsers that support redirection can ignore this content and simply perform the redirection as specified in the header:

```
HTTP/1.1 301 Moved Permanently

Location: http://www.somewhere-else.com/davepage.html

Content-Type: text/html

<HTML>

          <HEAD> <TITLE>Dave’s Not Here, Man!</TITLE> </HEAD>

                  <BODY BGCOLOR="#ffffff">

                  <h3>Dave’s Not Here, Man!</h3> Dave is no longer at this URL. If you want to visit
                  him, click <A HREF="http://www.somewhere-else.com/davepage.html">here</A>

          </BODY>

</HTML>
```

This response should cause the browser to generate the following request:

```
GET /davepage.html HTTP/1.1

Host: www.somewhere-else.com
```

The difference between the 301 and 302 status codes is the notion of ‘moved permanently’ versus ‘moved temporarily’. The 301 status code informs the browser that the data at the requested URL is now permanently located at the new URL, and thus the browser should always automatically go to the new location. In order to make this happen, browsers need to provide a persistence mechanism for storing relocation URLs. In fact, the mechanism used for storing cookies, authorization credentials, and cached content can be employed for this purpose as well. In the future, whenever a browser encounters a request for a relocated URL, it would automatically build a request asking for the new URL.