Direccionamiento y HTTP

Revisión: 2025-05-21

Direccionamiento

E12. En la siguiente URI `https://tools.ietf.org/html/rfc2616#section-14.19` ¿Cómo se llama la parte "`section-14.19`"? ¿Se envía por la red al hacer el request? ¿Por qué? Piense usos.

E13. ¿Cuál es la diferencia entre una URI y una URI reference?

E14. Dada la URI base `http://a/b/c/d;p?q` resuelva las referencias enumeradas en la sección 5.4.1. del [RFC3986]

E15. ¿Según la interpretación que tiene HTTP sobre las URIs: las siguientes URL son equivalentes?

a. `http://abc.com:80/~smith/home.html`
b. `http://ABC.com/%7Esmith/home.htm`
c. `http://ABC.com:/%7esmith/home.htm`

E16. En el siguiente elemento XML ¿Cuales de las siguientes URL tienes sentido que sean propiamente URL y cuales pueden ser URNs?

```
<foo xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
     xmlns:util="http://www.springframework.org/schema/util"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/springbeans.xsd
     http://www.springframework.org/schema/util
     http://www.springframework.org/schema/util/springutil.xsd">
```

netcat

E17. Comunique dos terminales utilizando **netcat**.

E18. {W} HTTP/1.1 y otros protocolos utilizan las secuencias `CR LF` como marcas de fin de línea. ¿Como usted puede lograr enviar esta secuencia con netcat desde una terminal donde se consumen caracteres desde la entrada estándar?

E19. {W} Con netcat escuche en el puerto TCP 9090. Ingrese la URL `http://localhost:9090/` en un *User Agent*:

a. Analizar detalladamente la estructura del request y en particular los headers que el user agent envió
b. Desde el server (netcat) retorne una respuesta que no siga las reglas de `HTTP`. ¿Qué sucede en el *user agent*?

c. Vuelva a realizar el request y desde el server (netcat) retorne una respuesta de formato válido. Verifique comportamiento en el *user agent*.

E20. {W} Con netcat conéctese a `www.google.com.ar` al puerto 80, y envíe de header `GET / HTTP/1.1\r\n\r\n` (sin ningún otro header).

a. ¿Qué código de respuesta retorna el servidor? ¿Qué significa? ¿Qué header es requerido que esté presente en dicho código?

b. Realizar los cambios necesarios en el request para que el servidor envíe como representación a nuestro pedido el formulario de búsqueda (es un html).

c. Analizar todos los headers que aparecen en la respuesta y su impacto en los UA (Ejemplo: `Date`, `Cache-Control`, `Content-Type`, `Set-Cookie`).

E21. Investigue el comando **curl** y el comando **wget**. ¿En que situación utilizaría cada una?. Preste atención a las opciones `-0 --http1.1 --http2 --compressed -d -H -i -s --socks5` del comando **curl**.

E22. {W} Con la utilidad wget descargue un archivo "*grande*". En el medio de la descarga cancele la misma; y luego intente resumir la descarga (`--continue`).

¿Qué headers participan para resumir las descargas?

¿Funciona con todas las páginas? Si no funciona; busque un contra ejemplo.

HTTP: Funcionalidades que provee el protocolo

E23. Los siguientes headers HTTP... ¿Son equivalentes? `Accept: text/plain` y `AccEpT: text/plain`

E24. 1. ¿Qué significa en la respuesta del servidor un header `Transfer-Encoding: chunked`?

2. ¿Qué otras codificaciones existen?

3. ¿Para que sirven?

4. ¿En que ocasiones utilizaría cada uno?

E25. a. ¿Qué es un media type?

b. ¿Para que se usa?

c. ¿Cómo está estructurado?

d. ¿Cuales son los media type asociados a …?

i. Páginas web (ej: html)

ii. Archivos de texto plano

iii. Estilos de páginas webs (CSS)

iv. Javascript (archivos de extensión .js)

v. Imágenes jpeg, gif, png

vi. archivos pdf

vii. un archivo .exe o de formato no estandar

E26. a. ¿Cuales son los requisitos para que un cliente pueda utilizar el esquema de conexiones persistentes de HTTP?
b. ¿Qué ventajas brinda frente al esquema de pedidos tradicional?
c. ¿De qué se trata el pipelining?

E27. a. ¿Qué métodos provee HTTP?
b. ¿Qué significa que un método es idempotente?
c. Piense del efecto de la idempotencia en el siguiente caso:

> Se cuenta con una extensión al user-agent (ej: plugin de Firefox) que una vez cargada las página web (de formato HTML) el mismo recolecta todos los links en el documento, y con el objetivo de acelerar la experiencia del usuario, comienza a descargar todos los links recolectados. De esta forma el usuario cuando siga algún link de interés no tendrá que esperar. Suponga que el usuario visita un sitio web que tiene una página web que presenta un listado de cosas, y cada ítem, además de presentar su nombre, presenta un link para editar su contenido, y otro link para eliminar el mismo (es decir el clásico ABM). El desarrollador decidió que realizando un GET a los link de eliminación (`/items/123/delete`).

- ¿Qué sucederá cuando el usuario que tiene el acelerador de internet activado acceda al ABM?
- ¿Quién tiene la culpa de ese comportamiento?
- ¿Como se podría solucionar el problema?

d. ¿Por qué se dice que `DELETE` es idempotente si es que el método borra el recurso de la misma forma que el comando **rm** borra un archivo?
e. ¿Cuando utilizaría `POST`? (Además del RFC, revise [whenToUseGet])

E28. Utilice el método `TRACE` contra www.google.com.ar para detectar si hay un proxy entre su computadora y el servidor de Google. Si ocurre algún error elabore una hipótesis de por que sucede.

E29. ¿Que código de retorno utilizaría desde su aplicación web para las siguientes situaciones?
a. El recurso al que se está accediendo ya no se encuentra en la dirección a la que se está intentando acceder. Se conoce cual es la nueva dirección que se debe utilizar a partir de este momento.
b. El cliente no tiene permiso para acceder a la página (ejemplo un usuario anónimo encontró cual es la URL del backoffice de administración)
c. El cliente no se encuentra autenticado
d. El recurso que se está editando (por ejemplo una entrada de un blog) fue editado desde el momento que nosotros lo comenzamos a editar, y apretamos el botón guardar (no queremos perder los cambios recién guardados)
e. ¿Cuando usaría el código 410? ¿En qué se diferencia en su opinión del 404?
f. Faltó en la URL de un `GET` un parámetro (ej `/foo/bar?param=123`)

g. El formato de un parámetro es incorrecto (ej `/foo/bar?param=abc` donde `param` debía ser un número entero).
h. Se está intentando subir un recurso cuyo formato no es soportado en el servidor
i. El recurso al que se está intentando acceder requiere de otro sistema externo que no se encuentra por el momento disponible (ej: está caído).
j. El cliente está enviando un `DELETE` al recurso pero esa operación no es soportada (solo soporta el `GET`).

E30. Si se encuentra snifeando la red y se encuentra con el siguiente pedido:

```
GET / HTTP/1.1
Authorization: Basic YWxndW51c3VhcmlvOmFsZ3VuYXBhc3N3b3Jk
```

a. ¿De qué se trata el header `Authorization`?
b. ¿Podría recuperarse el usuario y la password? Si es así...¿cuales son?

E31. ¿Qué estrategias provee HTTP para minimizar el tráfico de red y la utilización de recursos del servidor?

HTTP: Entity tags, y caching

E32. {W} Con la herramienta **cURL**, y el recurso `http://protos.foo/` responda las siguientes preguntas:
a. Haga un `GET` condicional utilizando la fecha de última modificación
b. Haga un `GET` condicional utilizando los "*entity tags*"

E33. ¿Qué interpretará un UA que tiene soporte de caching si recibe en una respuesta el header `Cache-Control: max-age=3600, must-revalidate`?

E34. Si hablamos de *Shallow Etag*... ¿De qué hablamos? [deep-tag]

E35. {W} Acceda utilizando Google Chrome a https://campus.itba.edu.ar/.

a. ¿Cómo se envía los datos del formulario? ¿Cómo está codificado?
b. Una vez que está logueado haga click en la materia Protocolos de Comunicación. Si HTTP busca no tener estado...¿Como hace el servidor para saber que es usted el que está haciendo el pedido?
c. Haga el pedido a esta URL usando el cURL ¿Visualiza el mismo contenido que en el browser? ¿Por qué?
d. Utilizando cURL, obtenga el HTML visualiza el desde el browser.

Nota

Debido a que HTTP se trasporta utilizando TLS si utiliza un sniffer no podrá analizar el tráfico. Sin embargo puede utilizar las *Developers Tools* (**F12**/ Network) para analizarlo.

HTTP: Configuración de servidor http, virtual hosts

Instalar un servidor nginx.

E36. El servidor proveerá hosting para el sitio web `foo` y para el sitio `bar`.

- Cuando un usuario se conecta a `foo` debe ver el mensaje "*Bienvenido a Foo*"
- Cuando un usuario se conecta a `bar` debe ver el mensaje "*Bienvenido a Bar*"
- Cuando un usuario se conecta con cualquier otro nombre al servidor (como ser utilizando sólo la dirección `IP`) el usuario debe ver el mensaje "*What?*"

Aviso

Podría encontrarse con un problema al acceder http://foo/ desde un user agent avanzado (como Chrome o Firefox). Investigue a que se debe.

E37. ¿Como prueba que el servidor está bien configurado aún cuando su computadora no resuelva los nombre `foo` y `bar`?

¡Comprobarlo!

Luego haga los cambios necesarios en el archivo `/etc/hosts` [hosts(5)] para poder acceder con el navegador.

E38. {W} Analizando los headers de pedido y respuesta, investigar las diferencias que hay al acceder desde un browser a un recurso versus recargar dicho recurso. Para la recarga discriminar entre el uso de la tecla F5 y la combinación de teclas `Ctrl+F5` (en Firefox).

E39. Proxy reverso:

a. Inicie un servidor HTTP sencillo utilizando python:

```
mkdir -p /tmp/protos
cd /tmp/protos/
echo "Hola mundo" > index.html
python3 -m http.server 8080
```

b. Verificar que está corriendo `http://localhost:8080/`

c. Haga los cambios necesarios en el nginx para que cuando acceda al host `foo` ve el contenido servido por Python

d. ¿Qué ventajas piensa que tiene éste esquema de despliegue?

> A "gateway" (a.k.a. "reverse proxy") is an intermediary that acts as an origin server for the outbound connection but translates received requests and forwards them inbound to another server or servers. Gateways are often used to encapsulate legacy or untrusted information services, to im-

> prove server performance through "accelerator" caching, and to enable partitioning or load balancing of HTTP services across multiple machines.
>
> —RFC7230 2.3 Intermediaries

E40. En el sitio `foo` y en el `bar` configure el nginx para que comprima de forma transparente los recursos que sirve.

a. ¿Que ventajas tiene este esquema?

b. ¿Afecta la performance?

c. ¿Salva ancho de banda y aumenta positivamente la experiencia de usuario?

d. `foo` estaba sirviendo contenido que era traído desde tomcat. Vuelva a pensar sobre las ventajas de éste esquema de despliegue.

E41. Para el sitio `foo` configure el nginx para que acepte conexiones del tipo SSL.

a. Utilice mkcert para generar una PKI (Public Key Infrastructure) local.

```
wget "https://dl.filippo.io/mkcert/latest?for=linux/amd64" -O /tmp/
mkcert
sudo mv /tmp/mkcert /usr/local/bin/mkcert
sudo chmod +x /usr/local/bin/mkcert
mkcert -install
```

b. Con la PKI instalada, genere un certificado para el dominio `foo`.

```
mkcert foo
```

c. Instale el certificado actualizando la configuración de su servidor HTTP proveyendo las rutas de los archivos generados. Recuerde diferenciar entre el componente **público** y el **privado** del certificado.

```
...
ssl_certificate /tmp/foo.pem;
ssl_certificate_key /tmp/foo-key.pem;
...
```

d. Compruebe lo que sucede al acceder desde un UA al sitio `https://foo`.

Lecturas recomendadas

[DesignIssues] *Axioms of Web Architecture: 0 [https://www.w3.org/DesignIssues/Model]* . Tim Berners-Lee. World Wide Web Consortium. January 1998.

[Addressing] *Naming and Addressing: URIs, URLs, ... [https://www.w3.org/Addressing/]* . Tim Berners-Lee. World Wide Web Consortium. 1993.

[URI-Style] *Cool URIs don't change [https://www.w3.org/Provider/Style/URI]* . Tim Berners-Lee. World Wide Web Consortium. 1998.

[CURIE-S] *CURIE Syntax 1.0: A syntax for expressing Compact URIs [https://www.w3.org/TR/curie/]* . W3C Working Group Note. World Wide Web Consortium. 16 December 2010.

[selfDescribingDocuments] *The Self-Describing Web [https://www.w3.org/2001/tag/doc/selfDescribingDocuments]* . . World Wide Web Consortium. 07 February 2009.

[http-perf96] *Analysis of HTTP Performance [http://www.isi.edu/touch/pubs/http-perf96/]* . Joe Touch, , y . Information Sciences Institute. Aug. 16, 1996.

We discuss the performance effects of using per-transaction TCP connections for HTTP access, and the proposed optimizations of avoiding per-transaction re-connection and TCP slow-start restart overheads. We analyze the performance penalties of the interaction of HTTP and TCP. Our observations indicate that the proposed optimizations do not substantially affect Web access for the vast majority of users, who typically see end-to-end latencies of 100-250 ms and use low bandwidth lines. Under these conditions, there are only 1-2 packets in transit between the client and server, and the optimizations reduce the overall transaction time by only 11%. Rates over 200 Kbps are required in order to achieve at least a 50% reduction in transaction time, resulting in a user-noticeable performance enhancement.

[whenToUseGet] *URIs, Addressability, and the use of HTTP GET and POST [http://www.w3.org/2001/tag/doc/whenToUseGet.html]* . . World Wide Web Consortium. 21 March 2004.

An important principle of Web architecture is that all important resources be identifiable by URI. The finding discusses the relationship between the URI addressability of a resource and the choice between HTTP GET and POST methods with HTTP URIs. HTTP GET promotes URI addressability so, designers should adopt it for safe operations such as simple queries. POST is appropriate for other types of applications where a user request has the potential to change the state of the resource (or of related resources). The finding explains how to choose between HTTP GET and POST for an application taking into account architectural, security, and practical considerations.

This finding does not discuss URI schemes other than "http" or protocols other than HTTP/1.1 [RFC2616].

[RFC2046] *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types [https://tools.ietf.org/html/rfc2046]* . N. Freed y N. Borenstein. The Internet Engineering Task Force. November 1996.

This second document defines the general structure of the MIME media typing system and defines an initial set of media types.

[RFC7230] *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing [https://tools.ietf.org/html/rfc7230]* . R. Fielding y J. Reschke. The Internet Engineering Task Force. June 2014.

The Hypertext Transfer Protocol (HTTP) is a stateless application- level protocol for distributed, collaborative, hypertext information systems. This document provides an over-

view of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes related security concerns for implementations.

[RFC3987] *Internationalized Resource Identifiers (IRIs) [https://tools.ietf.org/html/rfc3987]* . M. Duerst y M. Suignard. The Internet Engineering Task Force. January 2005.

This document defines a new protocol element, the Internationalized Resource Identifier (IRI), as a complement of the Uniform Resource Identifier (URI). An IRI is a sequence of characters from the Universal Character Set (Unicode/ISO 10646). A mapping from IRIs to URIs is defined, which means that IRIs can be used instead of URIs, where appropriate, to identify resources. The approach of defining a new protocol element was chosen instead of extending or changing the definition of URIs. This was done in order to allow a clear distinction and to avoid incompatibilities with existing software. Guidelines are provided for the use and deployment of IRIs in various protocols, formats, and software components that currently deal with URIs.

[RFC3986] *Uniform Resource Identifier (URI): Generic Syntax [https://tools.ietf.org/html/rfc3986]* . Roy Fielding y Larry Masinter. The Internet Engineering Task Force. January 2005.

[RFC6266] *Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP) [https://tools.ietf.org/html/rfc6266]* . Reschke, J.. The Internet Engineering Task Force. June 2011.

RFC 2616 defines the Content-Disposition response header field, but points out that it is not part of the HTTP/1.1 Standard. This specification takes over the definition and registration of Content-Disposition, as used in HTTP, and clarifies internationalization aspects. [STANDARDS-TRACK]

[RFC6585] *Additional HTTP Status Codes [https://tools.ietf.org/html/rfc6585]* . M. Nottingham y R. Fielding. The Internet Engineering Task Force. April 2012.

This document specifies additional HyperText Transfer Protocol (HTTP) status codes for a variety of common situations. [STANDARDS-TRACK]

[RFC7540] *Hypertext Transfer Protocol Version 2 (HTTP/2) [https://tools.ietf.org/html/rfc7540]* . Mike Belshe y Roberto Peon. The Internet Engineering Task Force. May 2015.

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unsolicited push of representations from servers to clients. This specification is an alternative to, but does not obsolete, the HTTP/1.1 message syntax. HTTP's existing semantics remain unchanged.

[mnot_cache] *Caching Tutorial for Web Authors and Webmasters [https://www.mnot.net/cache_docs/]* . Mark Nottingham. 6 May, 2013.

This is an informational document. Although technical in nature, it attempts to make the concepts involved understandable and applicable in real-world situations. Because of this, some aspects of the material are simplified or omitted, for the sake of clarity. If you are interested in the minutia of the subject, please explore the References and Further Information at the end.X

[deep-tag] *REST Tip: Deep etags give you more benefits [https://web.archive.org/web/20080220005452/https://bitworking.org/news/150/REST-Tip-Deep-etags-give-you-more-benefits]* . Joe Gregorio. 2007-03-22.

ETags, or entity-tags, are an important part of HTTP, being a critical part of caching, and also used in "conditional" requests. So what is an etag?

[REST] *Architectural Styles and the Design of Network-based Software Architectures [http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm]* . Roy Fielding. University of California, Irvine. 2000.

The World Wide Web has succeeded in large part because its software architecture has been designed to meet the needs of an Internet-scale distributed hypermedia system. The Web has been iteratively developed over the past ten years through a series of modifications to the standards that define its architecture. In order to identify those aspects of the Web that needed improvement and avoid undesirable modifications, a model for the modern Web architecture was needed to guide its design, definition, and deployment.

Software architecture research investigates methods for determining how best to partition a system, how components identify and communicate with each other, how information is communicated, how elements of a system can evolve independently, and how all of the above can be described using formal and informal notations. My work is motivated by the desire to understand and evaluate the architectural design of network-based application software through principled use of architectural constraints, thereby obtaining the functional, performance, and social properties desired of an architecture. An architectural style is a named, coordinated set of architectural constraints.

This dissertation defines a framework for understanding software architecture via architectural styles and demonstrates how styles can be used to guide the architectural design of network-based application software. A survey of architectural styles for network-based applications is used to classify styles according to the architectural properties they induce on an architecture for distributed hypermedia. I then introduce the Representational State Transfer (REST) architectural style and describe how REST has been used to guide the design and development of the architecture for the modern Web.

REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. I describe the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, contrasting them to the constraints of other architectural styles. Finally, I describe the lessons learned from applying REST to the design of the Hypertext Transfer

Protocol and Uniform Resource Identifier standards, and from their subsequent deployment in Web client and server software.

[URI-2001] *URIs, URLs, and URNs: Clarifications and Recommendations 1.0 [https://www.w3.org/TR/uri-clarification/]* . W3C/IETF URI Planning Interest Group. 21 September 2001.

This paper addresses and attempts to clarify two issues pertaining to URIs, and presents recommendations. Section 1 addresses how URI space is partitioned and the relationship between URIs, URLs, and URNs. Section 2 describes how URI schemes and URN namespace ids are registered. Section 3 mentions additional unresolved issues not considered by this paper and section 4 presents recommendations.

[LinkedDataTutorial] *How to Publish Linked Data on the Web [http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/]* . 2007-07-11.

[LinkedData] *Linked Data [https://www.w3.org/DesignIssues/LinkedData.html]* . World Wide Web Consortium. 2006-07-27.

[LinkedData2] *Linked Data [https://www.w3.org/standards/semanticweb/data]* . World Wide Web Consortium.

[nginx] *nginx documentation [http://nginx.org/en/docs/]* . nginx.org.

[ReverseProxy] *Nginx/ReverseProxy [https://help.ubuntu.com/community/Nginx/ReverseProxy]* . nginx.org.

[nc(1)] *nc — arbitrary TCP and UDP connections and listens*. BSD General Commands Manual. February 7, 2012.

The nc (or netcat) utility is used for just about anything under the sun involving TCP, UDP, or UNIX-domain sockets. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scan# ning, and deal with both IPv4 and IPv6. Unlike telnet(1), nc scripts nicely, and separates error messages onto standard error instead of send# ing them to standard output, as telnet(1) does with some.

[curl(1)] *curl - transfer a URL*. November 30, 2014.

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction.

curl offers a busload of useful tricks like proxy support, user authen# tication, FTP upload, HTTP post, SSL connections, cookies, file trans# fer resume, Metalink, and more. As you will see below, the number of features will make your head spin!

[wget(1)] *Wget - The non-interactive network downloader*. June 14, 2016.

GNU Wget is a free utility for non-interactive download of files from the Web. It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies.

Wget is non-interactive, meaning that it can work in the background, while the user is not logged on. This allows you to start a retrieval and disconnect from the system, letting Wget finish the work. By contrast, most of the Web browsers require constant user's presence, which can be a great hindrance when transferring a lot of data.

[hosts(5)] *hosts - static table lookup for hostnames*. Linux Programmer's Manual. March 29, 2015.

This manual page describes the format of the /etc/hosts file. This file is a simple text file that associates IP addresses with hostnames, one line per IP address.

[mkcert] *A simple zero-config tool to make locally trusted development certificates with any names you'd like*. May 21, 2025.

mkcert is a simple tool for making locally-trusted development certificates. Open source, code available in GitHub [https://github.com/FiloSottile/mkcert]

Using certificates from real certificate authorities (CAs) for development can be dangerous or impossible (for hosts like example.test, localhost or 127.0.0.1), but self-signed certificates cause trust errors. Managing your own CA is the best solution, but usually involves arcane commands, specialized knowledge and manual steps.

mkcert automatically creates and installs a local CA in the system root store, and generates locally-trusted certificates. mkcert does not automatically configure servers to use the certificates, though, that's up to you.