Tecnologie Web

Un **ipertesto** è un insieme di documenti messi in relazione tra loro tramite collegamenti monodirezionali (*hyperlink*). Può essere visto come una rete e i documenti ne costituiscono i nodi: attraverso un link possiamo passare da un punto di un documento all'altro. La lettura può svolgersi in maniera non lineare: qualsiasi documento della rete può essere il successivo. *Sono possibili infiniti percorsi di lettura.*

Il **World Wide Web** è un ipertesto distribuito sulla rete: i documenti, chiamati pagine, risiedono su server geograficamente distribuiti (*World Wide*) e costituiscono una ragnatela virtuale (*Web*). Da un qualunque documento è possibile saltare ad un altro indipendentemente da dove questo si trovi. La navigazione è detta browsing o surfing.

Elementi costitutivi del Web

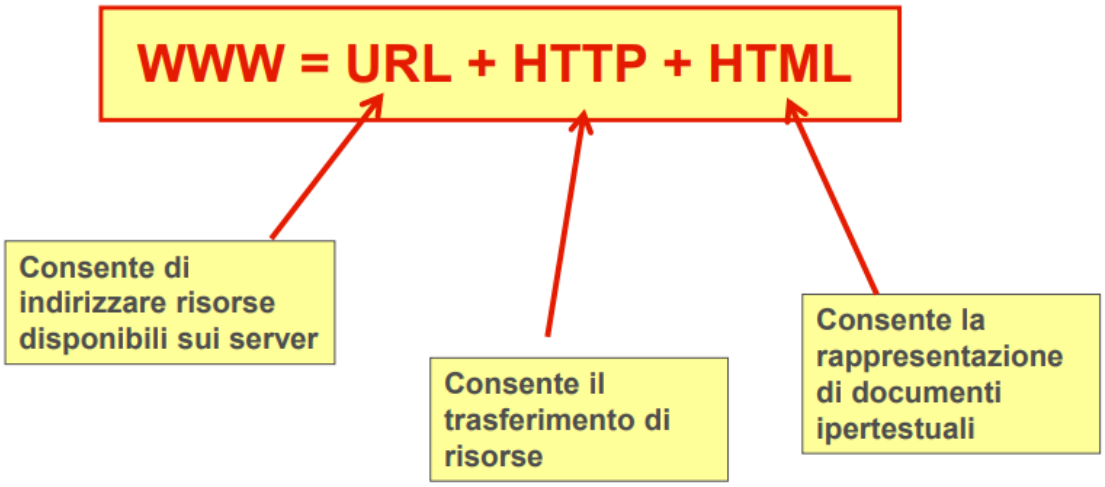
L'idea di base è la semplicità. Per realizzare questo ipertesto planetario abbiamo bisogno di 3 elementi concettuali:

1. Un meccanismo per localizzare un documento.
2. Un protocollo per accedere alle risorse che costituiscono il documento e trasferirle al client.
3. Un linguaggio per descrivere i documenti ipertestuali.

E due elementi fisici:

1. Un **server** in grado di erogare le risorse che costituiscono i documenti.
2. Un **client** in grado di rappresentare/visualizzare i documenti e di consentire la navigazione da un documento all'altro.

In estrema sintesi nella sua versione iniziale il Web può essere rappresentato con la “formula”:



Web segue un modello **Client/Server**:

- **Client ATTIVI**

  Detti anche Web Browser, richiedono pagine Web ai server e ne visualizzano semplicemente il contenuto.

  Utilizzano il protocollo HTTP per connettersi al server e URL per identificare risorse.

- **Client PASSIVI**

  Detti anche Web Server, forniscono ai client le pagine Web che questi richiedono.

  Rimangono in ascolto di eventuali connessioni di nuovi client e utilizzano il protocollo HTTP per interagire con i client.

URL: problematiche fondamentali

Il primo termine della “formula del Web” fa riferimento a 3 questioni principali:

1. Come identifichiamo il **server** in grado di fornirci un elemento dell’ipertesto?
2. Come identifichiamo la **risorsa** a cui vogliamo accedere?
3. Quali meccanismi possiamo utilizzare per accedere alla risorsa?

La risposta a tutte e 3 le domande sono gli **URI** (*Uniform Resource Identifier*), che forniscono un meccanismo semplice ed estendibile per identificare una risorsa. Si tratta di una sequenza di caratteri che identifica universalmente e univocamente una risorsa, non per-forza legata al Web e alla rete comunque.

La sequenza di caratteri è una stringa con una sintassi definita, dipendente dallo schema, che si presenta nella forma:

<scheme>:<scheme-specific-part>

Esiste un sottoinsieme di URI che condivide una sintassi comune per rappresentare relazioni gerarchiche in uno spazio di nomi:

<schema>://<authority><path>?<query>

Esistono due specializzazioni del concetto di URI:

- **Uniform Resource Name (URN)**

  Identifica una risorsa per mezzo di un “nome” in un particolare dominio di nomi, che deve essere globalmente unico e restare valido anche se la risorsa diventa non disponibile o cessa di esistere. Consente di parlare di una risorsa prescindendo dalla sua ubicazione e dalle modalità con cui è possibile accedervi.

  Si tratta quindi di un’identificazione concettuale.

  Ad esempio il codice ISBN dei libri.

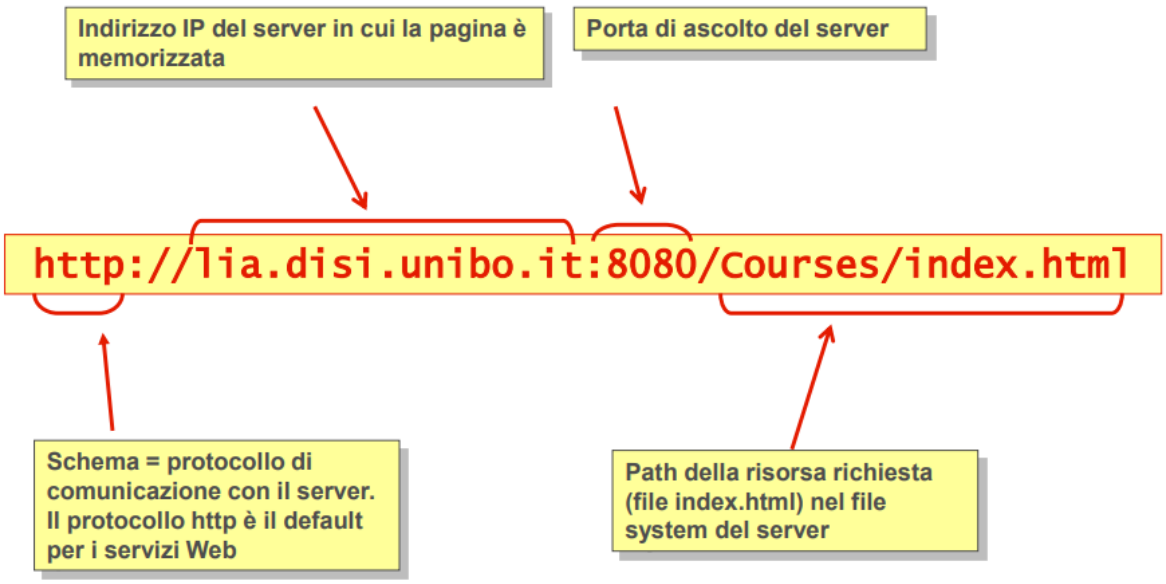
- **Uniform Resource Locator (URL)**

  Identifica una risorsa tramite la sua “collocazione” in un grafo. Identifica la risorsa non per il nome, ma per come la si può reperire.

Specifica il protocollo necessario per il trasferimento della risorsa stessa.

Tipicamente il nome dello schema corrisponde al nome del protocollo utilizzato. Nella forma più comune:

<protocol>://[<username>:<password>@]<host>[:<port>][/<path>[?<query>][#fragment]]



Gli URI possono essere classificati anche come:

- URI opaca: non soggetti a ulteriori operazioni di parsing
- URI gerarchica: è soggetta a ulteriori operazioni di parsing, per esempio per separare l'indirizzo del server dal percorso all'interno del file system.

Operazioni sulle URI gerarchiche

- **Normalizzazione**: processo di rimozione dei segmenti "." e ".." dal path di una URI gerarchica.
- **Risoluzione**: è il processo che a partire da una URI originaria porta all'ottenimento di una URI risultante.

- **Relativizzazione**: è il processo inverso alla risoluzione.

HTTP

HTTP è l'acronimo di HyperText Transfer Protocol. E' il protocollo di livello applicativo utilizzato per trasferire le risorse Web da server a client. Gestisce sia le richieste inviate al server che le risorse inviate al client. Inoltre è un protocollo ***stateless***: né il server né il client mantengono, a livello di protocollo, informazioni relative ai messaggi precedentemente scambiati.

Terminologia HTTP:

- **Client**: programma applicativo che stabilisce una connessione al fine di inviare delle richieste.
- **Server**: programma applicativo che accetta connessioni al fine di ricevere richieste ed inviare specifiche risposte con le risorse richieste.
- **Connessione**: circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione.
- **Messaggio**: è l'unita base di comunicazione HTTP, definita come una specifica sequenza di byte concettualmente atomica.
- **Resource**: oggetto di tipo dato univocamente definito.
- **URI**: identificatore unico per una risorsa.
- **Entity**: rappresentazione di una risorsa che può essere incapsulata in un messaggio (*tipicamente*) di risposta.

Questo protocollo è basato su TCP: sia le richieste al server che le risposte ai client sono tramesse usando uno stream TCP. In pratica:

1. Il server rimane in ascolto *tipicamente sulla porta 80.*
2. Il client apre una connessione TCP sulla porta 80.

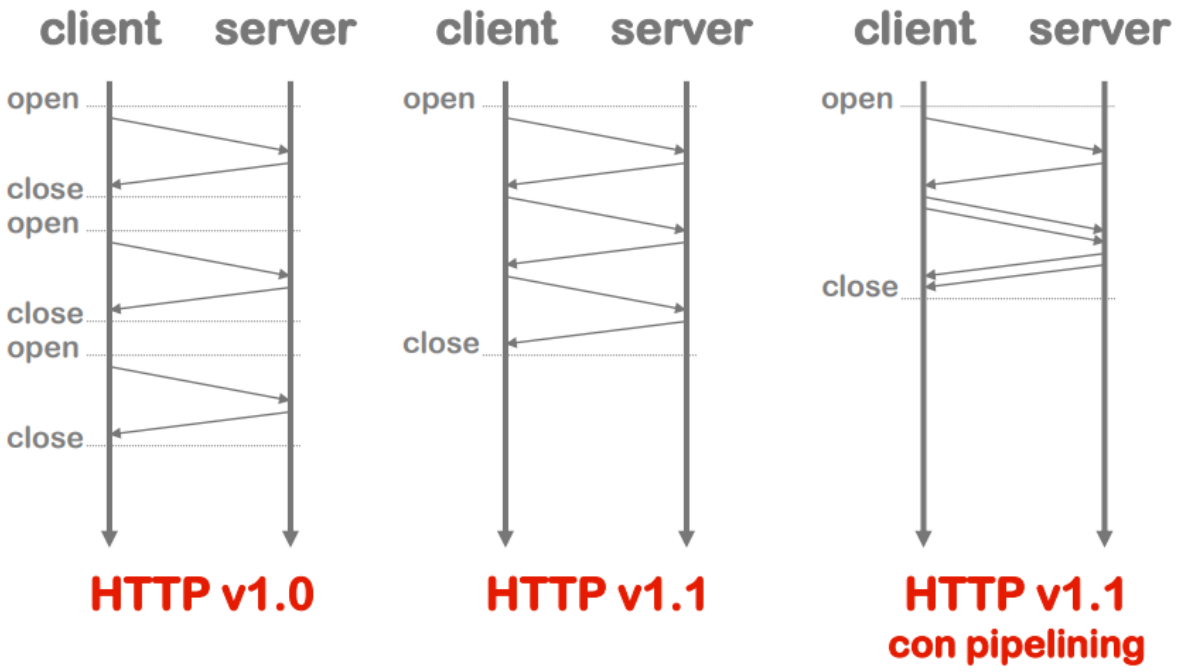
3. Il server accetta la connessione.
4. Il client manda una richiesta.
5. Il server invia una risposta e chiude la connessione.

Per questo diciamo anche che la versione HTTP v1.0 è ***oneshot*** nel senso che viene aperto e chiuso il canale TCP dopo ogni richiesta-risposta.

La differenza tra le versioni di HTTP, in particolare tra la v1.0 e la v1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione. In pratica il server, con la v1.1 lascia aperta la connessione TCP dopo aver spedito la risposta e piò quindi ricevere le richieste successive sulla stessa connessione.

Il server HTTP chiude la connessione quando viene specificato nell'header del messaggio oppure quado non è usata da un certo tempo.

Un ulteriore tecnica per migliorare le prestazioni è il **pipelining**, che consiste nell'invio di molteplici richieste da parte del client prima di terminare la ricezione della risposte. Le risposte in questo caso devono essere date nello stesso ordine delle richieste, poiché non è specificato un modo esplicito di associazione tra richiesta e risposta.



- HTTP v1.0 : 1 richiesta e 1 risposta per ogni connessione.
- HTTP v1.1 : più richieste per ogni connessione.
- HTTP v1.1 con pipelining: più richieste anche contemporaneamente per ogni connessione.

HTTP

Un messaggio HTTP è definito da due strutture:

- **Message Header**: contiene tutte le informazioni necessarie per identificare il messaggio.
- **Message Body**: contiene i dati trasportati dal messaggio.

Gli header sono costituiti da insieme di coppie `nome : valore` che specificano le caratteristiche del messaggio tramesso o ricevuto.

Esempio di Request

```
GET /search?q=Introduction+to+XML+and+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Chrome/38.0 (X11; U; Linux i686; en-US; rv:1.7.2)
   Gecko/20040803
Accept: text/xml,application/xml,application/xhtml+xml,
   text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: da,en-us;q=0.8,en;q=0.5,sw;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/
```

I comandi della richiesta

- **GET**: serve per richiedere una risorsa ad un server. E' il metodo più frequente, che viene attivato facendo click sul link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo del browser.
- **POST**: progettato come messaggio per richiedere una risorsa. A differenza di Get, i dettagli per identificazione ed elaborazione della risorsa stessa non sono nell'URL ma invece direttamente nel body message. NON ci sono quindi limiti di lunghezza nei parametri.
- **PUT**: chiede la memorizzazione sul server di una risorsa all'URL specificato. Serve quindi a trasmettere delle informazioni dal client al server. A differenza di Post si ha la creazione (*o sostituzione se già esisteva*) della risorsa.
- **DELETE**: richiede la cancellazione della risorsa riferita dall'URL specificato.

- **HEAD**: molto simile a Get, ma il server deve risponde solo con gli header relativi, senza includere il body. Viene dunque usato per verificare gli URL sia a livello di validità (*la risorsa esiste e ≠ 0*) che a livello di accessibilità (*non è richiesta la connessione*).
- **OPTIONS**: serve per richiedere informazioni sulle opzioni disponibili per la comunicazione.
- **TRACE**: è usato per invocare il loop-back remoto a livello applicativo del messaggio richiesta. *consente cioè al client di vedere cosa è stato ricevuto dal server.*

Esempio di Response

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 2008 12:00:15 GMT
Server: Apache/2.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2008 …...
Content-Length: 6821
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.51 Transitional//EN">
<html>...</html>
```

Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica. Ci sono 5 classi:

1. 1xx: **Informational**. Una risposta temporanea alla richiesta durante il suo svolgimento.
2. 2xx: **Seccesful**. Il server ha ricevuto, capito ed accettato la richiesta.
3. 3xx: **Redirection**. Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la risposta

4. 4xx: **Client error**. La richiesta del client non può essere soddisfatta per n errore da parte del client (*anche risorsa non autorizzata*)
5. 5xx: **Server error**. La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno.

Cookie

Parallelamente alle sequenze Request / Response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e viceversa: i **cookie**. Questi possono essere generati da ambo le parti e, dopo la creazione, vengono passati ad ogni trasmissione di request e responde. Il loro scopo è quello di fornire un supporto per il *mantenimento di stato* in un protocolo come HTTP che è essenzialmente ***stateless***.

In pratica sono una collezione di stringhe:

- **Key**: identifica univocamente un cookie all'interno del dominio.
- **Value**: valore associato al cookie (*stringa di max 255 caratteri*).
- **Path**: stabilisce la posizione nell'albero di un sito al quale è associato.
- **Domain**: dominio dove è stato generato.
- **Max-age**: numero di secondi di vita (*permette ad esempio la scadenza di una sessione*).
- **Secure**: stabilisce che questi cookie vengono trasferiti soltanto se il protocollo è sicuro (*HTTPS*).
- **Version**: identifica la versione del protocollo di gestione dei cookie.

Autenticazione

Esistono situazioni in cui si vuole restringere l'accesso alle risorse ai soli utenti abilitati. Diverse sono le tecniche comunemente utilizzate:

▼ **Filtro su set di indirizzi IP**

Questa soluzione presenta diversi svantaggi: non funziona se l'indirizzo è pubblico (NAT) e se l'indirizzo è assegnato dinamicamente (DHCP). Inoltre esistono varie tecniche che consentono di "presentarsi" con un indirizzo IP fasullo.

▼ **Form per la richiesta di Username e Password**

Analoghe considerazioni fatte con HTTP Basic, ma invece della GET viene utilizzata una Post nel dialogo Client —>Server

▼ **HTTP Basic**

Quando il Server vuole richiedere un'autenticazione può inviare una richiesta, utilizzando un codice 401 contenente un header HTTP chiamato WWW-Authenticate.

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Secure Area"
```

Quando il Client vuole inviare le proprie credenziali può utilizzare l'header HTTP Authorization così formato:

- Username e Password uniti nella stringa "username:password"
- Il risultato è codificato in base64
- Il metodo di autenticazione e uno spazio sono inseriti all'inizio della stringa codificata:

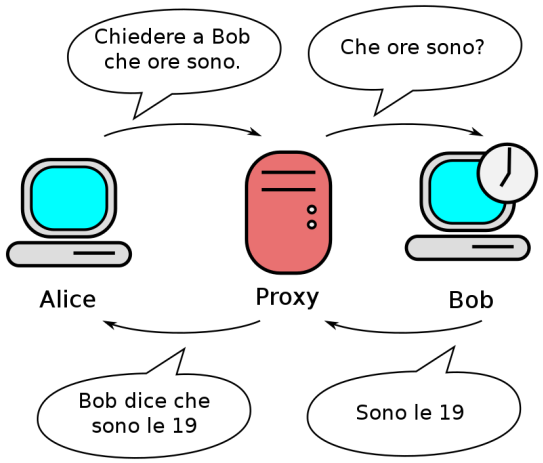
```
GET /private/index.html HTTP/1.1
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

▼ **HTTP Digest**

In disuso negli ultimi anni: consiste nell'invio di password dopo hash (*ha quindi maggior sicurezza rispetto alla semplice Base64 dell'HTTP Basic*)

Architetture più distribuite e articolate per il Web

- **Proxy**: programma applicativo in grado di agire sia come Client che come Server al fine di effettuare richieste per conto di altri Client. Le Request vengono processate internamente oppure vengono ridirezionate a Server. Un proxy deve interpretare e, se necessario, riscrivere le Request prima di inoltrarle. *un Client si connette al server proxy per richiedere un qualche servizio: il server proxy valuta ed esegue la richiesta di quest'ultimo in modo da semplificare e gestire la sua complessità.*



- **Gateway**: Server che agisce da intermediario per altri Server. Al contrario dei proxy, il gateway riceve le request come se fosse il server originale e i Client non sono in grado di identificare che Response proviene da un gateway. Nelle reti complesse con più subnet, ognuna fa riferimento a un gateway che si occuperà di instradare il traffico dati verso le altre sottoreti o reindirizzarlo ad altri gateway. *E' anche detto reverse-proxy*.
- **Tunnel**: programma applicativo che agisce come "bling relay" tra due connessioni. Una volta attivo non partecipa alla comunicazione HTTP.

Caching distribuito

L'idea di base è memorizzare copie temporanee di documenti Web al fine di ridurre l'uso della banda ed il carico sul Server. Una Web cache memorizza i documenti che la

attraversano: l’obiettivo è usare documenti in cache per le successive richieste qualora alcune condizioni siano verificate. Le **Web cache** possono essere:

▼ **User Agent Cache**

Lo User Agent (*tipicamente i browser*) mantiene una cache delle pagine visitate dall’utente. L’uso delle User Agent Cache risultava molto utile in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga: tuttavia anche oggi è utile ad esempio per far lavorare gli utenti con connettività intermittente.

▼ **Proxy Cache**

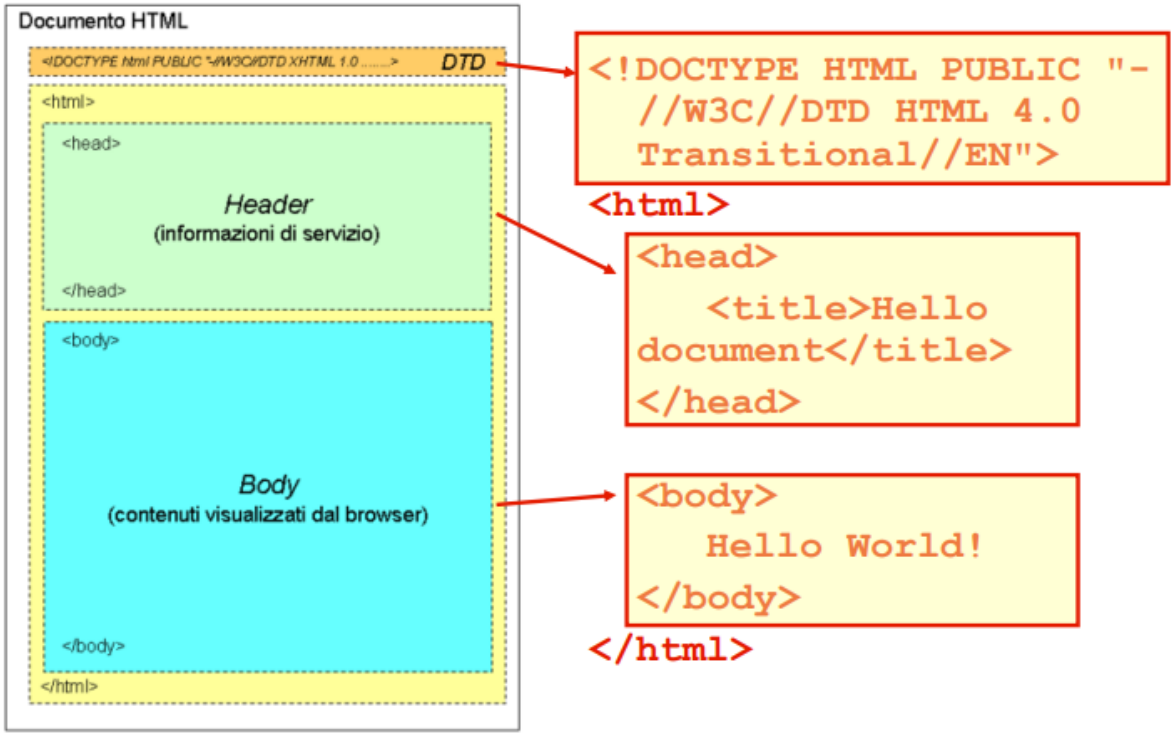
Servono per ridurre le necessità di banda: il proxy intercetta il traffico e mette in cache le pagine: successive richieste NON provocano lo scaricamento ulteriore delle pagine al server.

Ovviamente servono dei meccanismi che controllano il caching distribuito per evitare di incorrere in errori:

- **Freshness**: controlla il Time To Live dell’oggetto salvato in cache. Se vogliamo accedere alla risorsa e il TTL NON è ancora scaduto possiamo accedere direttamente; altrimenti prima di ogni accesso è necessario un controllo per verificare che l’oggetto è ancora valido.
- **Validation**: viene usato per controllare se un elemento in cache è ancora corretto
- **Invalidation**: è un processo per il quale gli oggetti della cache vengono rimpiazzati o rimossi a seguito di altre Request.

HTML

HTML è l’acronimo di HyperTextMarkupLanguage. E’ il linguaggio utilizzato per descrivere le pagine che costruiscono i nodi dell’ipertesto.



i **tag HTML** sono usati per definire il **mark-up** di elementi HTML.

<p> *Testo di un paragrafo* <p>

start tag Contenuto dell'elemento end tag

Un elemento può essere dettagliato mediante **attributi**. Gli attributi sono **coppie "nome=valore"** contenute nello start tag con una sintassi del tipo:

<tag attrib1='valore1' attrib2='valore2'>

apici singoli o doppi sono indifferenti

Header

E' identificato dal tag **<head>**. Contiene elementi non visualizzati dal browser.

```
<head>
    <meta http-equiv="Content-Type"
     content="text/html;
     charset=iso-8859-1">
    <meta name="description"
     content="Documentation about HTML">
    <meta name="keywords"
     content="HTML, tags, commands">
    <title>Impariamo l'HTML</title>
    <link href="style.css"
     rel=stylesheet type="text/css">
</head>
```

▼ **title**

Titolo della pagina (*nella testata principale*)

▼ **meta**

esistono due tipi di meta, distibnguibili dal primo attributo: http-equiv o name

Gli elementi di tipo http-equiv danno informazioni al browser su come gestire la pagina. Gli elementi di tipo name forniscono informazioni utili ma non critiche.

- **refresh**: http-equiv=refresh content=45 —> ogni 45t refresha la pagina

- **expires**: http-equiv=expires content="Tue, 20 Aug 2022 14:25:27 GMT" —> stabilisce la data di scadenza (*validità*) del documento
- **content type**: http-equiv="Content-Type" content="text/html ; charset=iso-8859-1" —> definisce il tipo di dati contenuto nella pagina
- **author**: name=author content="riccardo" —> specifica il nome dell'autore della pagina
- **description**: name=description content="home page unibo" —> descrizione della pagina
- **copyright**: name=copyright content="Copyright 2008, ricc" —> indica che la pagina è protetta da un diritto d'autore
- **keywords**: name=keywords lang="en" content="computer documentation" —> lista di parole chiave separate da virgole, usate dai motori di ricerca
- **date**: name="date" content="2008-05-07T09:10:56+00:00" —> data di creazione del documento

▼ **base**

Definisce come vengono gestiti i riferimenti relativi nei link (*la base URL*)

▼ **link**

Collegamento verso file esterni: CSS, script, icone visualizzabili nella barra degli indirizzi del browser

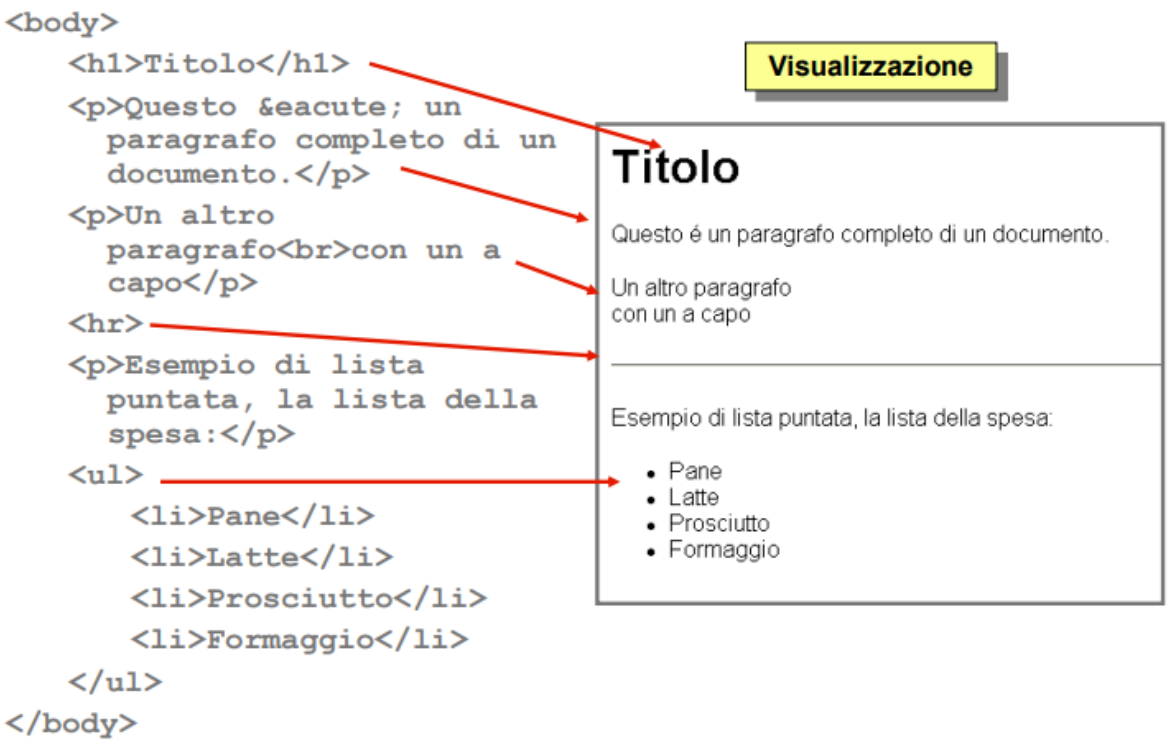
▼ **script**

Codice eseguibile utilizzato dal documento

▼ **style**

Informazioni di stile (CSS)

Body



Il body può contenere vari elementi:

- **Intestazioni**: titoli organizzati in gerarchia
- **Strutture di testo**: paragrafi, testo indentato, ecc.
- **Aspetto del testo**: grassetto, corsivo, ecc.
- **Elenchi e liste**: numerati, puntati, ecc.
- **Tabelle**
- **Form**: campi di inserimento, checkbox, menù a tendina, ecc.
- **Collegamenti ipertestuali**
- I**mmagini e contenuti multimediali**
- **Contenuti interattivi**: script, applicazioni esterne, ecc.

Dal punto di vista del layout della pagina, gli elementi HTML si dividono in 3 grandi categorie:

- **Elementi “block-level”**: costituiscono un blocco attorno a sè, e di conseguenza “vanno a capo”. *ad esempio paragrafi, tabelle ecc.* Un elemento di questo tipo può contenere altri elementi block-level/ inline.
- **Elementi “inline”**: NON vanno a capo e possono essere integrati nel testo. *ad esempio link, immagini, ecc.* Un elemento di questo tipo può contenere sono altri elementi inline.
- **Liste**: numerate, puntate…

HTML

Html, Css

Web Dinamico

Servlet

JSP

JavaScript

Ajax

React

Esame