Tecnologie Web L-A

Architetture e Protocolli per il Web

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Outline

- Il protocollo HTTP
  - Descrizione del protocollo
  - Gestione delle connessioni HTTP 1.0 vs HTTP 1.1
  - Cenni sulla sicurezza in HTTP
  - Gestione delle cache
- La notazione URL
- Modelli per Applicazioni Web
  - Programmazione Client-Side
  - Programmazione Server-Side
  - Web Proxy
  - Architetture Multi-Tier
- Semplici applicazioni client-server in Java basate sul protocollo HTTP

Breve Storia del Web

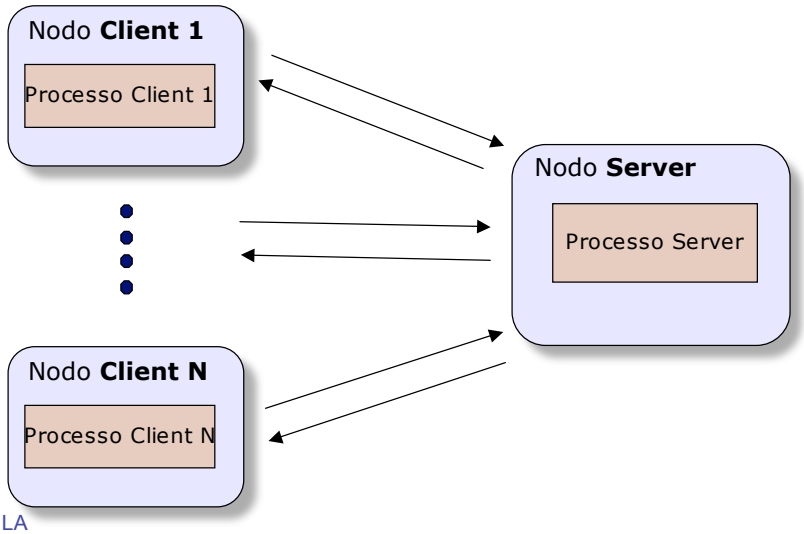
- Il **Word Wide Web** (WWW) è stato proposto nel **1989** da **Tim Berners-Lee**
- L'idea alla base del progetto era quella di fornire strumenti adatti alla **condivisione di documenti statici** in forma **ipertestuale** disponibili su **Internet** (rimpiazzare i sistemi basati su FTP)
- In estrema sintesi possiamo dire che nella sua formulazione iniziale:

Modello del Web

- Il **Web** segue un modello **Client/Server**
- I Client
  - utilizzano il protocollo **http** per **connettersi** ai **server**
  - **Richiedono pagine web** ai **server** e nel **visualizzano** il **contenuto**
  - I client sono **tipicamente web browser**, es IE, Mozilla. Si stanno però **diffondendo** di **client differenti** es. Apple widget, MS gadgets, etc.
- I Server
  - **Rimangono** in **ascolto** di eventuali **connessioni** di nuovi **client**
  - **Utilizzano** il protocollo **http** per interagire con i client
  - **Forniscono** ai client **le pagine web** che questi richiedono

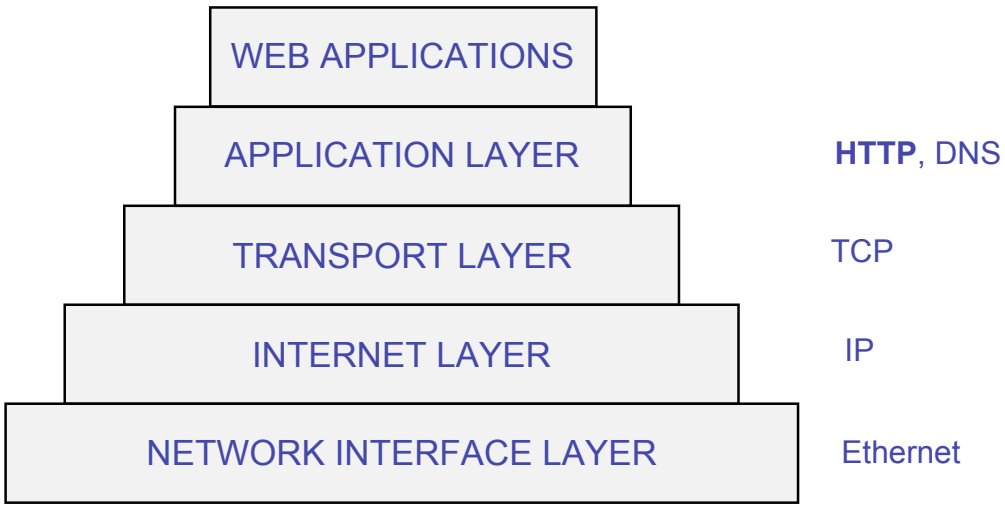
Modello Client/Server

- È un modello di **comunicazione asimmetrica** (molti:1)
- Il **Cliente designa** esplicitamente il **destinatario**
- Il **Servitore risponde** al **processo** che ha **effettuato** una **richiesta**



HTTP: Hyper Text Transfer Protocol

- Protocollo basato su TCP
- Sia richieste al server, sia le risposte ai client sono trasmesse usando stream TCP



Principali Questioni

- Come identifichiamo il Server di nostro interesse?
- Come identifichiamo la risorsa a cui vogliamo accedere?
- Quali meccanismi possiamo utilizzare per accedere alla risorsa?

Ovviamente queste sono solo le prime domande a cui dobbiamo trovare risposta. E' evidente che ulteriori questioni devono essere affrontate: sicurezza, ottimizzazione dei sistemi nella erogazione dei servizi web, etc. etc.

Uniform Resource Identifier

- Forniscono un **meccanismo semplice** ed **estensibile** per **identificare** una **risorsa**
- Per **risorsa** intendiamo qualunque cosa che abbia una **identità**. Esempi sono un documento, una immagine, un servizio, una collezione di risorse.
  - Non tutte le risorse sono disponibili in rete
  - **Mapping concettuale** fra una entità (o un insieme di entità) e non necessariamente alla entità che corrisponde al mapping in un dato momento. Di conseguenza il **mapping** ad una risorsa **può rimanere inalterato anche se cambia** il **contenuto** della **risorsa**

Uniform Resource Identifier

- L'**identificatore** è un oggetto che **riferisce** una entità che ha una identità. Nel caso della URI è una sequenza di caratteri che rispetta una precisa sintassi.
- L'accesso **uniforme** ha diversi vantaggi
  - **Differenti tipologie** di identificatori possono essere **usati** nello **stesso contesto indipendentemente** dal **meccanismo** di **accesso**
  - Stabilisce una **comune semantica** per l'interpretazione
  - Stabilisce **medesime convenzioni sintattiche**
  - **Facilita** l'**introduzione** di **nuovi tipi** di **identificatori** per le risorse

URI, URN ed URL

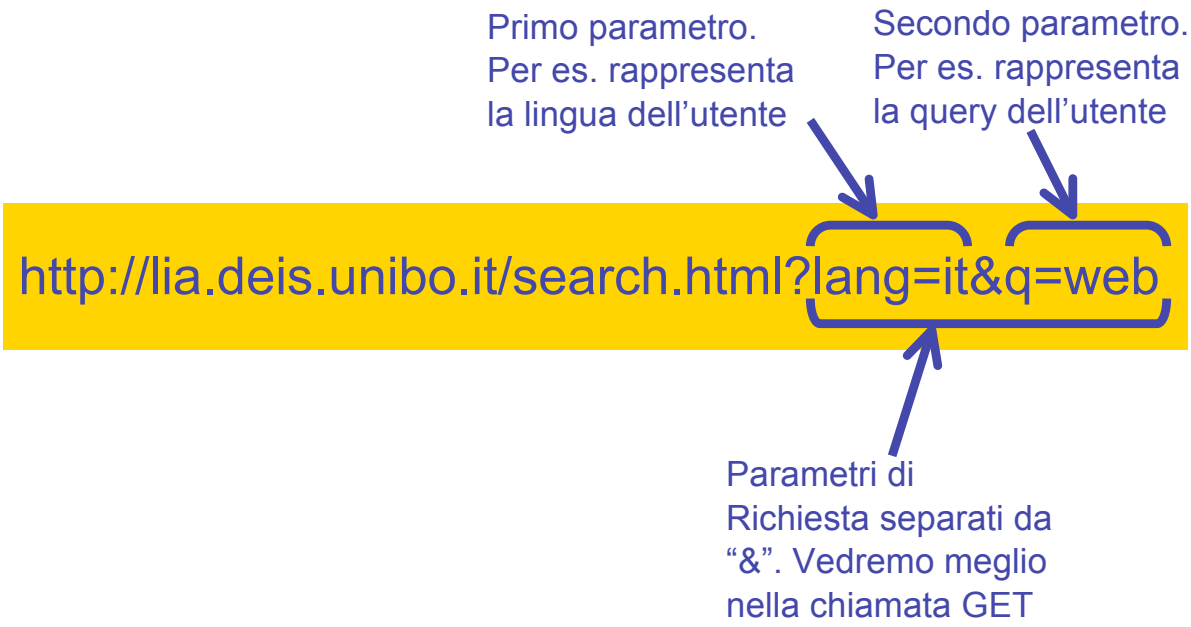
- Le URI possono essere classificate in:
  - **Uniform Resource Locator** (URL)
    - Il termine URL riferisce il **sottoinsieme** delle URI che **identificano** le **risorse** per mezzo del loro **meccanismo** di **accesso primario** (es. la loro locazione nella rete) piuttosto che sulla base del loro nome o attributi.
  - **Uniform Resource Name** (URN)
    - Il termine URN riferisce il **sottoinsieme** delle URI che devono rimanere **globalmente uniche** e **persistenti anche qualora** la **risorsa cessi** di **esistere** o diventi **non disponibile**.
    - Esempio **urn:isbn:0-395-36341-1** stabilisce il **sistema** di **identificazione** International Standard Book Number e l'**idenficatore del libro** ma **non** dice **come ottenerne** una **copia**.

Uniform Resource Locator

- **Uniform Resource Locator**: rappresenta l'**estensione** dell'**URI tenendo conto** del **protocollo** necessario per il trasferimento della risorsa. Per il protocollo HTTP l'URL è il seguente:
  - **http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]**
  - Il termine **URL** è informale, e **usato solo** per **taluni protocolli**, tra cui **HTTP**.
- Se la porta non viene specificata viene scelta la **porta 80** come da **default** dello **standard**
- **Se il path non viene specificato** interviene il **percorso** di **root del Web Server**
- La **chiave "?"** serve per la **specifica degli eventuali parametri** nella **richiesta** della risorsa (chiamata in get)

Uniform Resource Locator (URL)

Uniform Resource Locator (URL)



HTTP: Hyper Text Transfer Protocol Terminologia

- **Client**: Programma **applicativo** che stabilisce una **Connessione** al fine di **inviare** delle **Request**
- **Server**: Programma applicativo che **accetta Connessioni** al fine di **ricevere Request** ed **inviare specifiche Response** con le **risorse richieste**.
- **Connessione**: **circuito virtuale** stabilito a livello di **trasporto** tra due applicazioni **per fini di comunicazione**
- **Messaggio**: è l'**unità base** di **comunicazione HTTP**, è **definita** come una **specifica sequenza** di byte **concettualmente atomica**.
  - **Request**: messaggio HTTP di richiesta
  - **Response**: messaggio HTTP di risposta
  - **Resource**: Oggetto di tipo dato univocamente definito
  - **URI**: Uniform Resource Identifier – identificatore unico per una risorsa.
- **Entity**: **Rappresentazione di una Risorsa**, può essere incapsulata in un messaggio.

HTTP: Hyper Text Transfer Protocol Messaggio

- Un **messaggio HTTP** è definito da due strutture:
  - **Message Header**: Contiene tutte le informazioni necessarie per la identificazione del messaggio (più ingenerale tutte le intestazioni del messaggio)
  - **Message Body**: Contiene i dati trasportati dal messaggio.
- Esistono degli **schemi precisi** per ogni tipo di **messaggio** relativamente agli **header** ed ai **body**
- I messaggi di **Response contengono** i **dati** relativi alle **risorse richieste** (nel caso più semplice la pagina html)
- I **dati** sono **codificati** secondo il **formato specificato** nell'**header**, solitamente sono in formato **MIME** (Multipurpose Internet Mail Extensions); è **possibile utilizzare** anche il formato **ZIP**. Relativamente ai form HTML i content type usati sono: ap*plication/x-www-form-urlencoded* (default) e, nel caso di upload di file, *multipart/form-data*

Get e Post

- **GET**: **richiedo** una specifica **risorsa** attraverso un singolo **URL**. Posso passare diversi parametri, la **lunghezza massima** di un **URL** è **limitata**
- **POST**: **richiedo** una specifica **risorsa** evidenziando che il **body** del **messaggio contiene** i **dettagli** per la **identificazione** e la **elaborazione** della risorsa stessa: **non** ci sono **limiti** di **lunghezza** nei **parametri** di una richiesta

Ma non solo GET e POST

- **DELETE**: richiedo la **cancellazione** della **risorsa riferita** dall'**URL** specificato..
- **PUT**: richiedo che il **documento** allegato sia **memorizzato** all'**URL specificato**.

Questi metodi sono **tipicamente disabilitati** sui **server** disponibili in rete perché **non** è **generalmente desiderabile** dare ai **client** la **possibilità** di **inserire** o **eliminare** risorse sui server. In realtà in tempi molto recenti alcuni servizi web hanno iniziato a sfruttare pienamente questi meccanismi (RESTful Web Service).

Ma non solo GET e POST

- **OPTIONS**: rappresenta la **richiesta** di **informazioni** sulle **opzioni disponibili** per la **comunicazione**.
- **TRACE**: è usato per **invocare** il **loop-back remoto** a livello applicativo del messaggio di richiesta. Consente al **client** di **vedere** cosa è stato **ricevuto** dal **server** ed ha applicazione nella **diagnostica** e nel **testing** dei servizi web.
- **HEAD**: è simile al metodo GET. A seguito di una HEAD il **server restituisce** solo lo **header** del messaggio di risposta. HEAD trova applicazione nel determinare meta-informazioni sul documento richiesto, senza la necessità di trasferirlo.

Esempio di Richiesta HTTP

```
GET /search?q=Introduction+to+XML+and+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (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/
```

- Request line (methods: GET, POST, ...)
- Header lines
- Request body (empty here)

Esempio di Risposta HTTP

```
HTTP/1.1 200 OK
Date: Fri, 17 Sep 2009 07:59:01 GMT
Server: Apache/2.0.50 (Unix) mod_perl/1.99_10 Perl/v5.8.4
↳mod_ssl/2.0.50 OpenSSL/0.9.7d DAV/2 PHP/4.3.8 mod_bigwig/2.1-3
Last-Modified: Tue, 24 Feb 2009 08:32:26 GMT
ETag: "ec002-afa-fd67ba80"
Accept-Ranges: bytes
Content-Length: 2810
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>...</html>
```

- Status line
- Header lines
- Response body

Status Code

- **1xx Informational**
  - Da **http1.0** in poi **non devono essere usati** se non in condizioni di test.
- **2xx Success**
  - La richiesta del client è stata ricevuta, e processata con **successo**
- **3xx Redirection**
  - Il client deve **intraprendere ulteriori azioni** per completare la **request**. Se il metodo per la request era **get** o **head** allora **non** è **necessaria interazione** con l'**utente**.
- **4xx Client Error**
  - **Errore** nella request **da parte** del **client**
- **5xx Server Error**
  - Il **server** ha subito un **errore** nel processare una **request apparentemente valida**

Esempi di Status Code

- **200 OK**
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- **404 Not Found**
- **500 Internal Server Error**
- 503 Service Unavailable
- ...

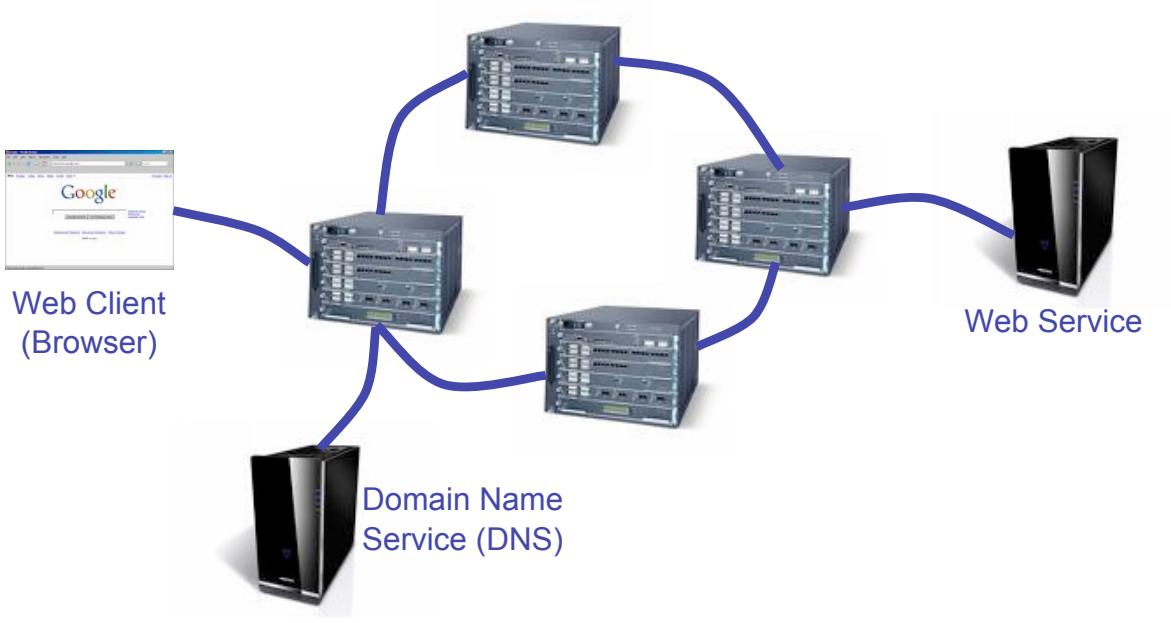
Esempi di Status Code

- **200 OK**
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- **404 Not Found**
- **500 Internal Server Error**
- 503 Service Unavailable
- ...

Alcune applicazioni web tendono a **non** restituire codici **404** ma a gestire l'errore a livello applicativo (**soft-404**). Se l'applicazione adotta la gestione soft-404 viene restituita una pagina con la stringa che segnala l'errore all'utente. La pagina è però mostrata all'utente come qualunque altra pagina e lo **status code** associato è **200**.

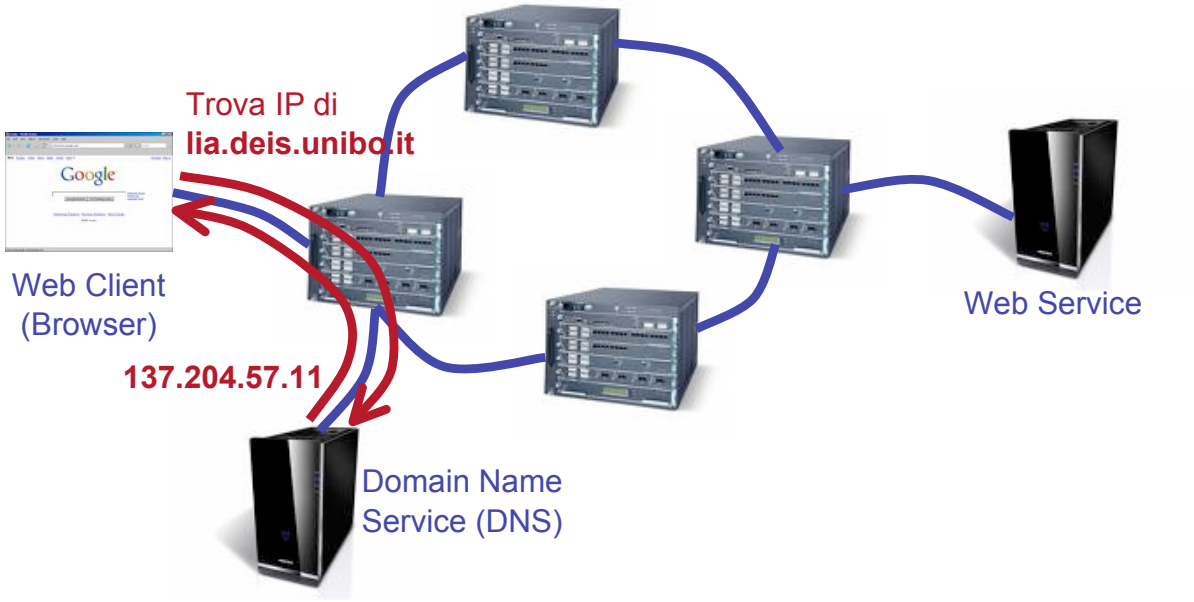
Un Semplice Esempio

Ipotizziamo di volere visitare il sito **lia.deis.unibo.it/Courses/TecWebLA/index.html**



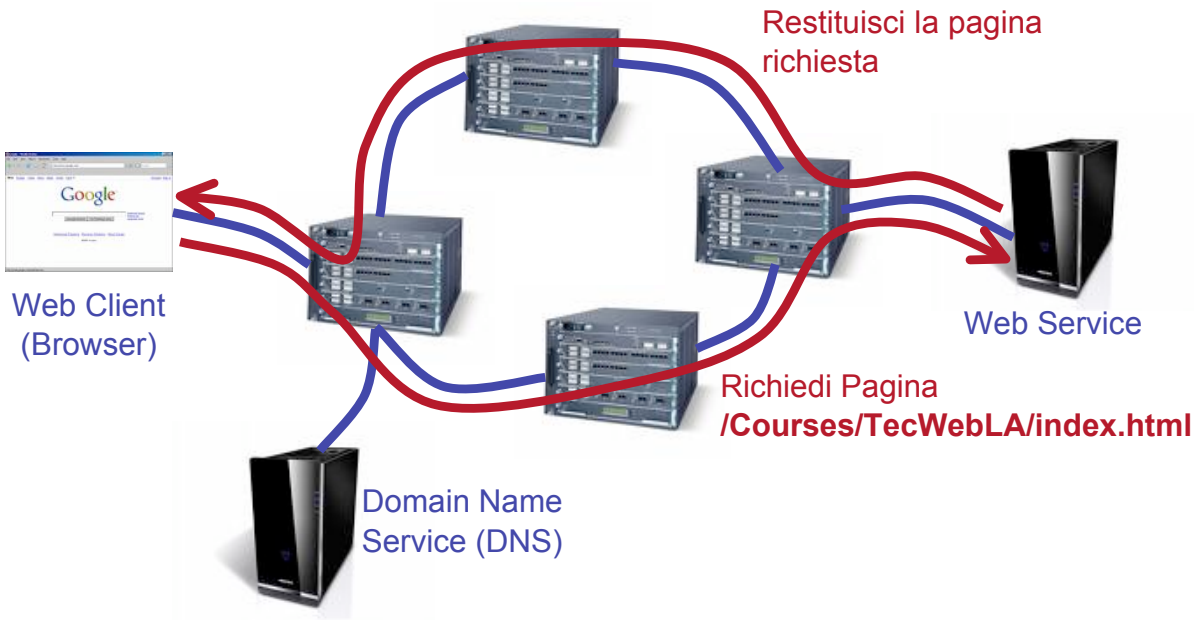
Un Semplice Esempio

1) Richiediamo al DNS l'indirizzo di **lia.deis.unibo.it**



Un Semplice Esempio

2) Richiediamo al server il file **/Courses/TecWebLA/index.html**



HTTP e TCP

- Viene usato **TCP** come protocollo di trasporto sottostante
- Il **client HTTP** dapprima **avvia** una **connessione TCP** con il **server**. I processi browser e server accedono al TCP tramite le loro **interfaccia socket**
- Sul lato client c'è la porta tra il processo client e la connessione TCP; lo stesso per il lato server
- Il **client** invia **messaggi** di **richiesta HTTP** tramite la sua **socket** e da questa **riceve messaggi** di **risposta** HTTP. **Lo stesso vale per il server**

HTTP e TCP

- **TCP** garantisce un **trasferimento affidabile** dei **messaggi** all'HTTP. HTTP non deve preoccuparsi dei dati persi o dei dettagli su come TCP ritrova e riordina i messaggi entro la rete.
- Il **server invia** al client i **file richiesti senza immagazzinare** alcuna informazione di **stato relativa al client**. Se il client chiede due volte lo stesso oggetto entro pochi secondi, il server lo rispedisce: HTTP è un protocollo senza stato (**stateless protocol**).

Connessione Non Permanente HTTP 1.0

Ipotizziamo di volere richiedere una pagina composta da un file HTML e 10 immagini JPEG

www. someSchool.edu/somedepartment/home.html

1. Il **client** inizia una **connessione TCP** con il **server** www.someSchool.edu sulla **porta 80**
2. Il client **invia** un messaggio di **richiesta** HTTP (GET) al server attraverso la **socket**. Il messaggio di richiesta richiede il nome del percorso (**/somedepartment/home.html**)
3. Il **server riceve** il messaggio attraverso la **socket** tcp stabilita, **trova** l'oggetto richiesto, lo **incapsula** in un **messaggio** HTTP e lo **restituisce** al client

Connessione Non Permanente HTTP 1.0

4. Il **server** HTTP **richiede** al TCP la **conclusione** della **connessione**
5. Il **client** HTTP **riceve** il **messaggio** di risposta e la **connessione** TCP si **conclude**. Il **messaggio indica** che l'oggetto incapsulato è in formato **HTML**.
6. Il **client estrae** il file dal messaggio, lo analizza e **trova** i **riferimenti** ai 10 oggetti JPEG
7. **Per ogni oggetto** trovato il **client richiede** al **server** di inviarlo utilizzando **HTTP**.

Ciascuna connessione TCP trasporta esattamente un messaggio di richiesta ed un messaggio di risposta. Nell'esempio si generano 11 connessioni TCP

Connessione Permanente HTTP 1.1

- Il **server lascia aperta** la **connessione** TCP dopo aver spedito la risposta. Le **successive richieste** e **risposte** sugli **stessi client** e **server** possono essere **inviate** sulla **stessa connessione**.
- Nell'esempio precedente **l'intera pagina** web (file HTML e 10 immagini) possono essere **inviate** sulla **stessa connessione** TCP permanente.
- Il **server** HTTP **chiude** la **connessione quando non** è **usata** da un certo tempo (intervallo di **time out**), che è tipicamente **configurabile**.

Connessione Permanente HTTP 1.1

- Per migliorare ulteriormente le prestazioni si usa la tecnica del **pipelining**
  - **Invio** di **molteplici richieste** da parte del **client prima** di **ricevere** le **risposte**
  - Ridotto numero di pacchetti TCP/IP
  - Applicabile solo per richieste idempotenti (es. GET)
  - Supportato dai browser moderni

Autenticazione

- **Restringere** l'**accesso** alle risorse ai soli **utenti abilitati**

- Tecniche comunemente utilizzate
  - Indirizzo IP
  - Form per la richiesta di username e password
  - HTTP Basic
  - HTTP Digest

Autenticazione

- Basare **l'autenticazione** sull'indirizzo **IP** del **client** è una soluzione che presenta vari **svantaggi** ed è per questo usata molto poco
- L'autenticazione **HTTP Digest** è caduta in **disuso** negli ultimi anni
- Normalmente usiamo
  - **Form**
  - **HTTP Basic**

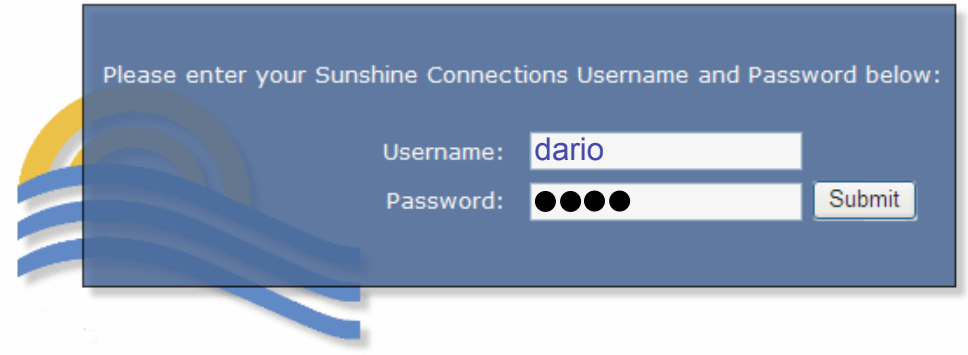
Autenticazione HTTP Basic

- *Challenge:*
  HTTP/1.1 401 Authorization Required
  WWW-Authenticate: Basic realm="The Doe Family Site"

- *Response:*
  Authorization: Basic emFjaGFyaWFzOmFwcGxlcGllCg==

Autenticazione Form

- Normalmente si usa il metodo **POST**
- Analoghe considerazioni a quelle fatte per HTTP Basic



Sicurezza

- Proprietà desiderabili
  - **Confidenzialità**
  - **Integrità** — **SSL/TSL**
  - **Autenticità**
  - Non Ripudio

- SSL: *Secure Sockets Layer*
- TLS: *Transport Layer Security*

SSL/TSL

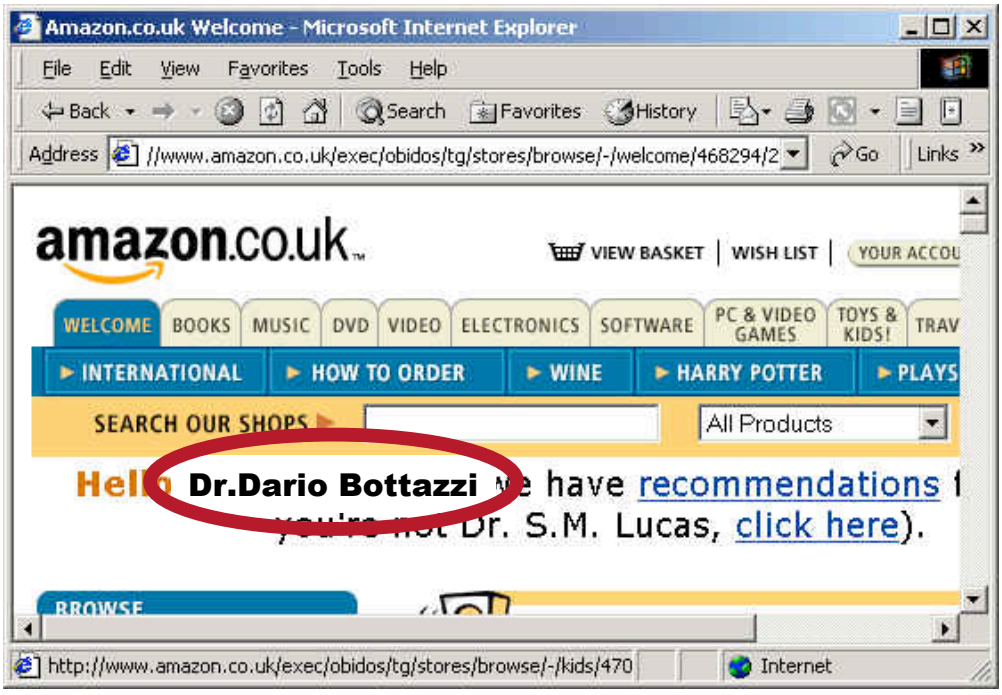
- Viene posto un **livello** che si occupa della gestione di confidenzialità, autenticità ed integrità della comunicazione **fra HTTP** e **TCP**
  - Accediamo tramite **https://...**

- Basato su **crittografia** a **chiave pubblica**
  - private key + public key
  - **certificato** (in genere usato per **autenticare** il **server**)

Sessione

- **HTTP** è un **protocollo Stateless** che **non** fornisce perciò **meccanismi** per la **gestione** della **sessione**
- **HTTP** vede **ogni richiesta** come **indipendente** e **stateless**
- Molte applicazioni però richiedono la **gestione** dello **stato**
  - *E-commerce* (shopping basket)
  - *Siti per utenti registrati*
- **Vari meccanismi** sono stati sviluppati **per** la **gestione** dello **stato** sulla sommità di **HTTP**

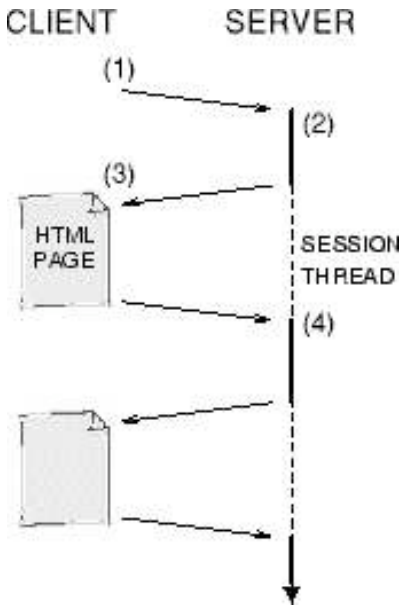
Esempio



Note sull'Esempio

- Anche se non ho visitato il sito per settimane, appena decido di accedervi vengo portato ad una pagina di benvenuto personalizzata
- Eccellente accorgimento per il business
- Risultato ottenuto tramite i cookies
- Funziona però solo se accedo sempre dalla stessa macchina

Tecniche per la Gestione della Sessione



- URL rewriting
- Hidden form fields
- Cookies
- SSL sessions

I Cookie

- **Informazioni testuali** passate negli **header HTTP**
- I **server inviano** i **cookies** nei messaggi di response
- Sono **memorizzati** e gestiti dal **browser**
- Il browser **invia** i **cookies** nelle future **request** allo **stesso server**
- **Possono essere cifrati** per non rivelare informazioni sensibili
- I cookies memorizzano la **chiave** per **l'accesso** ai **dati** di interesse e **non** i **dati stessi**
  - Per esempio un sito di e-commerce potrebbe memorizzare l'ID dello shopping basket dell'utente e non il suo contenuto
- I **cookies** hanno generalmente un **parametro *time to live*** che ne stabilisce il **periodo di validità**

I Cookie

- sono rappresentati da una **tupla** di **stringhe** con formato **attributo-valore**:
  - **Key**: identifica univocamente un cookie all'interno di un dominio:path
  - **Value**: valore associato al cookie (è una stringa di max 255 caratteri)
  - **Path**: posizione nell'albero di un sito al quale è associato (di default /)
  - **Domain**: dominio dove è stato generato
  - **Max-age**: (opzionale) numero di secondi di vita (permette la scadenza di una sessione)
  - **Secure**: (opzionale) non molto usato prevede una verifica di correttezza da parte del server
  - **Version**: identifica la versione del protocollo di gestione dei cookie

Esempio di Cookie

- Il client richiede un documento e riceve nel response

```
Set-Cookie: CUSTOMER=WILE_E_COYOTE;
path=/; expires=Wednesday, 09-Nov-99
23:12:40 GMT
```

Quando il client richiede una URL nel server viene spedito il cookie

```
Cookie: CUSTOMER=WILE_E_COYOTE
```

Per ulteriori dettagli sui cookie si consiglia di vedere il sito http://www.netscape.com/newsref/std/cookie_spec.html

Limitazioni dei Cookie

- **Non** sono **trasferibili** fra **browser diversi**
- **Non** sono solitamente **trasferibili** fra **macchine diverse**
- **Non** possono essere **riferiti** da una **URL**
- **Non** sono **sempre accettati**
  - Alcuni **browser** (per il vero pochi) **non** li **supportano**
  - Alcuni **utenti** preferiscono **disabilitare** i **cookies** per timori relativi alla **privacy**

URL Rewriting

- I **parametri** della **query** vengono **aggiunti** alla **URL**
  - Per esempio: botshop.it/basket?id=123
- Oppure il **session ID** potrebbe essere **inserito** nella **URL**
  - Per esempio: botshop.it/session125498/basket.jsp
- Normalmente si usa per **mantenere** il solo **session ID**. E' evidente che non ha molto senso memorizzare l'intero stato. Per i più scettici si ricordino limitazioni sulla lunghezza delle URL.
- **URL** possono essere **riferite** (**bookmark**) e se vogliamo **comunicate** ad altri (es. tramite mail)

Hidden Form Fields

- Alcuni autori preferiscono il termine **hidden form variables**
- Come vedremo diversi **elementi HTML** consentono di **acquisire input** dagli **utenti**
- Normalmente vogliamo rendere visibili questi elementi ma potremmo anche **renderli invisibili** ed **utilizzarli** per **tracciare** lo **stato** della **sessione**
- Per vederli dobbiamo consultare il sorgente
- Il **server** può **accedere** a questi **elementi** senza problemi. Es nelle JSP (che vedremo) possiamo usare `request.getParameter()`

SSL Sessions

- Alcuni autori parlano di **WWW-Authenticate**
- Applicabile per ambienti in cui si abbia un **sistema** di **autenticazione**
- Le connessioni **SSL** hanno una idea di sessione. Viene usato questo supporto per **gestire** lo **stato** della interazione con l'utente.
- Analogamente ai cookie le **informazioni** sono **passate** negli **header HTTP**.

Architetture Avanzate 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 **Clienti**. Le **Request** vengono **processate internamente oppure** vengono **ridirezionate** al Server. Un proxy deve **interpretare** e, **se necessario**, **riscrivere** le **Request** prima di inoltrarle
- **Gateway**: Server che **agisce** da **intermediario** per altri **Server**. Al contrario dei proxy, il **gateway riceve** le **request come se fosse** il **server** originale ed il **Client non** è in grado di **identificare** che la **Response** proviene da un **gateway.** Detto anche reverse proxy.
- **Tunnel**: Programma applicativo che **agisce** come "**blind relay**" tra due **connessioni**. Una volta attivo (in gergo "salito") **non partecipa** alla **comunicazione http**

Caching

- Idea di base: **memorizzare copie temporanee** di **documenti web** (es. pagine HTML, immagini) 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** i **documenti** in **cache** per le **successive richieste** qualora alcune **condizioni** siano **verificate**.
- Tipi di web cache
  - **User Agent Cache**
  - **Proxy Cache**

User Agent Cache

- Lo **user agent** (tipicamente il **browser**) **mantiene** una cache delle **pagine visitate** dall'**utente**.
- L'uso delle user agent cache era molto importante in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga
- Questo modello di caching è ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente. Nuovi strumenti, es. Google Gears, si basano su questo concetto.

Proxy Cache

- **Forward Proxy Caches**
  - Servono per ridurre le necessità di banda
  - Es. rete locale aziendale, Università, etc.
  - Il **proxy intercetta** il **traffico** e mette in **cache** le **pagine**
  - **Successive richieste non necessitano** di **richiedere** ulteriori copie delle pagine al **server**
- **Reverse Proxy Caches**
  - **Gateway cache**
  - **Operano** per **conto del server** e consentono di **ridurre** il **carico** computazionale delle macchine.
  - I **client non** sono in grado di **capire se** le **pagine arrivano** dal **server** o dal **gateway**
  - Internet Caching Protocol per il coordinamento fra diverse cache. Base per le content delivery networks.

HTTP e Cache

HTTP definisce vari meccanismi per la gestione delle cache

- **Freschness**: consente di **usare** una **response senza controllare** il **server**. Può essere usato da client e da server.
- **Validation**: può essere usato per **controllare** se un **elemento** in **cache** è **ancora corretto**, per esempio, nel caso in cui sia in cache da molto tempo
- **Invalidation**: è normalmente un **effetto collaterale** di altre request che hanno attraversato la cache. Se per esempio viene mandata una **POST**, una **PUT** o una **DELETE** ad una URL il **contenuto** della **cache** deve essere **invalidato**

La Ricerca di Informazioni su Web

- Il **Web** è un ipertesto (o un **grafo**) con **milioni** di **nodi**. È necessario **reperire** le **informazioni attraverso** i **link**
- Esistono indici del web (detti anche cataloghi o directories), realizzati per facilitare la ricerca di informazioni su Internet
- Organizzazione degli indici:
  - alfabetica
  - per argomento (gerarchici)
  - per area geografica (gerarchici)
  - con possibilità di ricerca (parole chiave)

La Ricerca di Informazioni su Web

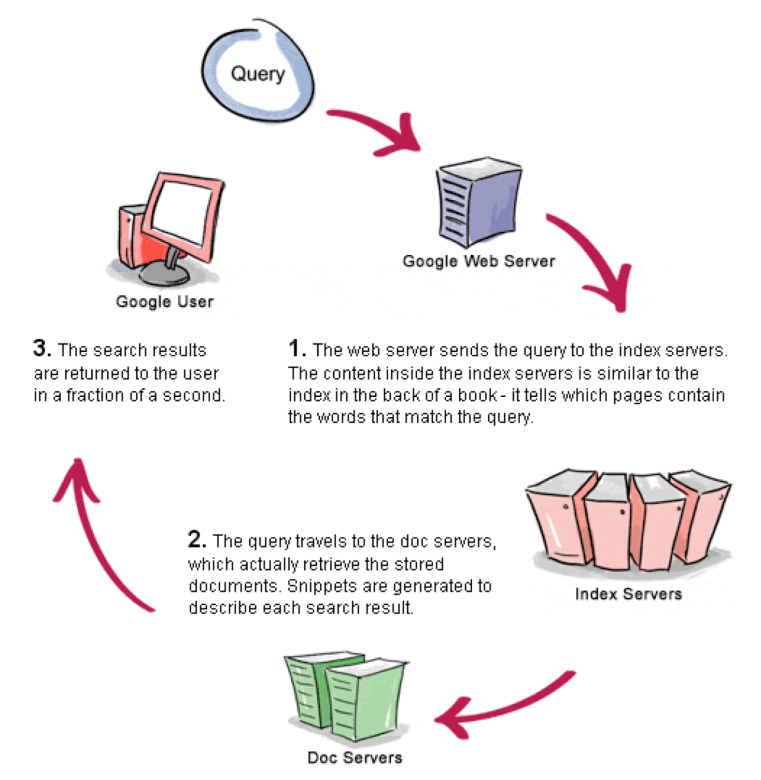
- Gli indici sono costruiti utilizzando dei programmi che esplorano i site web presenti in rete.
  - search engine
  - spider
  - crawler
  - worm
  - knowbots (knowledge robots)

Le Dimensioni del Sistema

Google (*www.google.com*), dati **2001**.

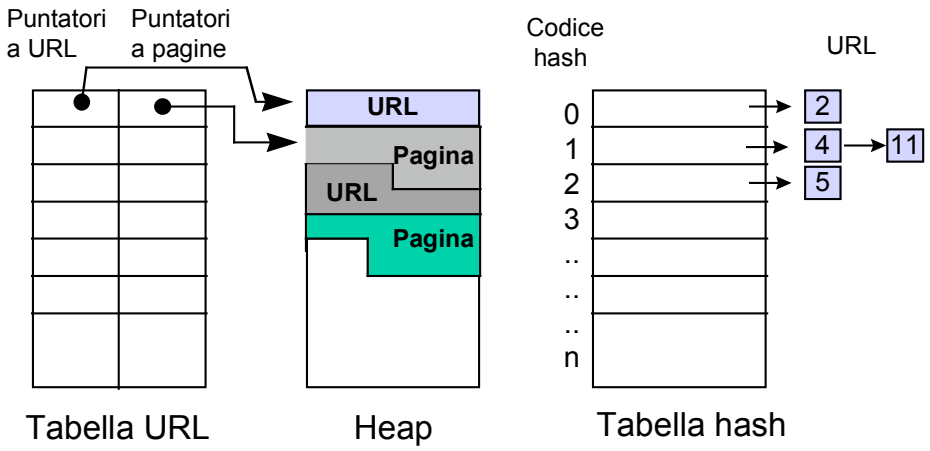
- Dati memorizzati:
  - indicizza 3 miliardi di pagine web
  - 700 milioni di messaggi usenet
- Statistiche di accesso:
  - 150 milioni di ricerche al giorno
- Architettura:
  - Cluster Linux di più di 10.000 macchine

Come Viene Processata una Query



I Motori di Ricerca

- Struttura di supporto tipica:



Motori di Ricerca

- I motori di ricerca devono soddisfare due principali requisiti
  - **Ricerca**: devono essere analizzate tutte le pagine web disponibili in rete
  - **Indicizzazione**: devono essere individuate le parole chiave all'interno delle pagine al fine di presentare all'utente le pagine di suo interesse

Fase di Ricerca

- Passi della ricerca (algoritmi **breadth-first**, **depth-first, random IP, random walk**):
  - **prelevare** una URL
  - **eseguire hash** URL
  - **Se** hash **URL** è in **Tabella hash** allora **STOP**
  - **altrimenti**
  - **aggiungere** hash **URL in Tabella hash**
  - **aggiungere Puntatori** a **URL** e a **pagina** in **Tabella**
  - **aggiungere URL** e **Pagina** (o titolo) in **Heap**
  - **ripetere** tutti i passi **per ogni link** della pagina

Problemi nella Ricerca

- **dimensioni** del **grafo** web e **mancanza** di **organizzazione** e **ordinamento** delle **pagine**
- **punto** di **partenza** della ricerca
- **peso dei link** (anche autorità e centralità)
  - Servono **algoritmi di ranking**
- **tipo di ricerca**:
  - depth-first -> stack overflow
  - breadth-first -> dimensioni heap
  - Metodi random -> problemi nella individuazione delle pagine

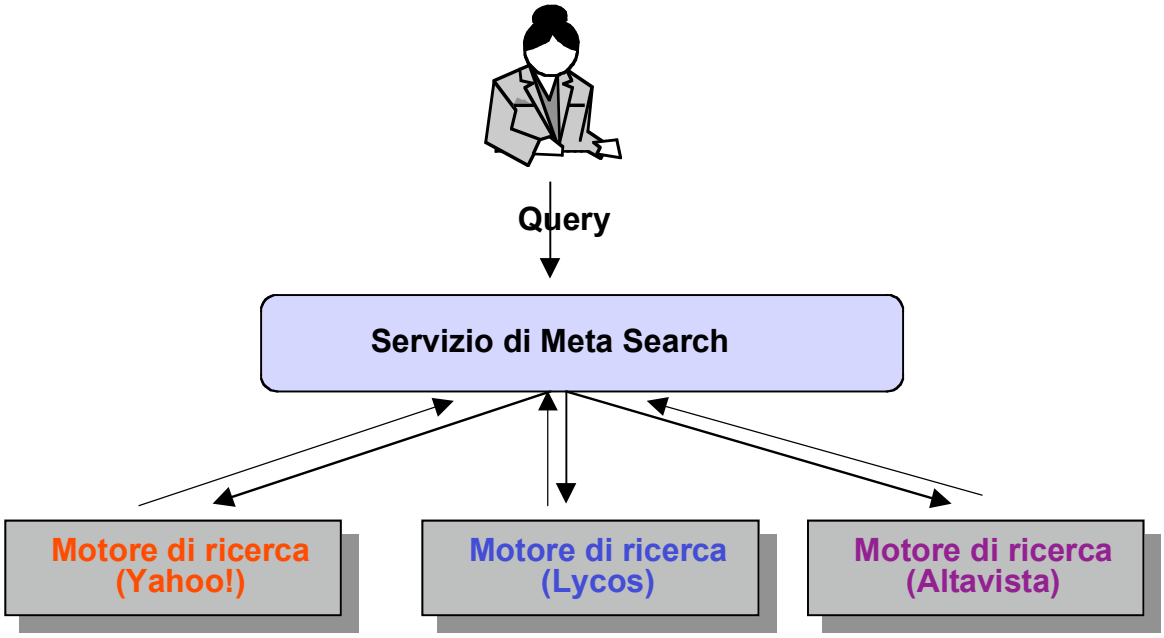
Fase di Indicizzazione

- La procedura di **indexing estrae** le **parole chiave** da ogni **pagina** (o titolo) web **memorizzati** nell'**heap** nella **fase di ricerca** (**sintesi delle pagine**)
- Per trovare le parole chiave:
  - si **scartano** le **parole poco significative** (articoli, etc.)
  - si scelgono **parole** che nella pagina hanno la **frequenza maggiore** (es. Lycos)
- Per ogni **parola** ottenuta si **memorizza** in una **tabella** la **parola** e **l'URL** che la **contiene**
- Alla fine dell'indicizzazione **si ordina** la **tabella** sulle **parole** e si **salva** su **file** che verrà **consultato** per le **ricerche** da parte degli **utenti**

Problemi di Indicizzazione

- **titoli pagine** spesso **poco significativi**
- **analisi** intere pagine **costosa**
- pagine solo **video** o **audio**, oppure in tecnologia **AJAX** possono essere **complesse** da indicizzare

I Meta-Search Engine



Robots.txt

- Standard per **limitare** l'accesso agli **spider** (detti anche robot) nell'**analisi** di un sito web. Lo **standard** è stato proposto e si è diffuso a partire dalla prima metà degli anni novanta

  http://www.robotstxt.org/orig.html
- Consente di **specificare** cosa **può** e cosa **non può** essere **analizzato** ed **indicizzato** da un **motore di ricerca**
- Data URL si scriva il file
  - URL/robots.txt
- Il file specifica le restrizioni dell'accesso

Esempio di Robots.txt

- **Nessun robot** dovrebbe **visitare** alcuna URL che parte con "**/botz/personal**", **a parte** il robot che si chiama "**botSearch**"

```
User-agent: *
Disallow: /botz/personal/

User-agent: botSearch
Disallow:
```


Programmazione Web con Java

- **Normalmente** i client per le applicazioni Web sono i **Browser**
- **Nessuno però ci obbliga** ad usare il browser e una tendenza rilevante degli ultimi anni è quella di scrivere **Rich-Internet Applications** (RIA)
  - Adobe AIR
  - Apple Widgets
  - …
- Queste applicazioni **normalmente** sono scritte in **javascript** ma possono essere implementate in linguaggi differenti, es. **Java**

TCP/IP: DomainName2IPNumbers

```
import java.net.*;

public class DomainName2IPNumbers {
  public static void main(String[] args) {
    try {
      InetAddress[] a = InetAddress.getAllByName(args[0]);
      for (int i = 0; i<a.length; i++)
        System.out.println(a[i].getHostAddress());
    } catch (UnknownHostException e) {
      System.out.println("Unknown host!");
    }
  }
}
```

```
java DomainName2IPNumbers www.google.com

66.102.9.104
66.102.9.99
```


TCP/IP: SimpleServer (1/2)

```
import java.net.*;
import java.io.*;

public class SimpleServer {
  public static void main(String[] args) {
    try {
      ServerSocket ss =
        new ServerSocket(Integer.parseInt(args[0]));
      while (true) {
        Socket con = ss.accept();
        InputStreamReader in =
          new InputStreamReader(con.getInputStream());
```

TCP/IP: SimpleServer (2/2)

```
      StringBuffer msg = new StringBuffer();
       int c;
       while ((c = in.read())!=0)
         msg.append((char)c);
       PrintWriter out =
         new PrintWriter(con.getOutputStream());
       out.print("Simon says: "+msg);
       out.flush();
       con.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```


TCP/IP: SimpleClient (1/2)

```
import java.net.*;
import java.io.*;

public class SimpleClient {
  public static void main(String[] args) {
    try {
      Socket con =
        new Socket(args[0], Integer.parseInt(args[1]));
      PrintStream out =
        new PrintStream(con.getOutputStream());
      out.print(args[2]);
      out.write(0);
      out.flush();
```

TCP/IP: SimpleClient (2/2)

```
    InputStreamReader in =
      new InputStreamReader(con.getInputStream());
     int c;
     while ((c = in.read())!=-1)
       System.out.print((char)c);
     con.close();
    } catch (IOException e) {
     e.printStackTrace();
    }
  }
}
```

```
java SimpleServer 1234
```

```
java SimpleClient localhost 1234 "Hello World"

Simon says: Hello World
```


I/O Non Bloccante

- Fornisce supporto per connessioni concorrenti e per buffering dei messaggi
- Packages: java.nio.channels, java.nio
- Classi principali:
  - ServerSocketChannel, SocketChannel
  - Selector
  - ByteBuffer
- Vedi API Java e Java Tutorial per maggiori dettagli

HTTP in Java

- Due approcci possibili
  - Si possono utilizzare le socket TCP e implementare a mano il protocollo HTTP. Scelta fortemente sconsigliata.
  - **Si possono usare** le classi di **Java** che consentono di lavorare con **HTTP**

Esempio Google I'm Feeling Lucky (1/2)

```
import java.net.*;
import java.io.*;

public class ImFeelingLucky2 {
  public static void main(String[] args) {
    try {
      String req = "http://www.google.com/search?"+
        "q="+URLEncoder.encode(args[0], "UTF8")+"&"+
        "btnI="+URLEncoder.encode("I'm Feeling Lucky", "UTF8");
      HttpURLConnection con =
        (HttpURLConnection) (new URL(req)).openConnection();
      con.setRequestProperty("User-Agent", "IXWT");
      con.setInstanceFollowRedirects(false);
```

Esempio Google I'm Feeling Lucky (2/2)

```
      String loc = con.getHeaderField("Location");
      System.out.print("The prophet spoke thus: ");
      if (loc!=null)
        System.out.println("Direct your browser to "+loc+
          " and you shall find great happiness in life.");
      else
        System.out.println("I am sorry - my crystal ball is blank.");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

```
java ImFeelingLucky2 W3C

The prophet spoke thus: Direct your browser to
http://www.w3.org/ and you shall find great
happiness in life.
```


RFC Downloader

Le RFC sono pubblicate sul sito

http://www.ietf.org

Le RFC sono identificate da numeri interi progressivi e sono reperibili in formato testuale alla url

http://www.ietf.org/rfc/rfcXXX.txt

Dove XXX è l'identificatore della RFC.

Si sviluppi un semplice applicativo per scaricare e leggere le RFC

Riferimenti

- A. Møeller, M. Schwartzbach, "Capitolo 3: il Protocollo HTTP", *Introduzione alle Tecnologie WEB*, Addison-Wesley, Gennaio 2006.
- RFC1945, "Hypertext Transfer Protocol - HTTP/1.0", *http://www.ietf.org/rfc/rfc1945.txt*
- RFC2616, "Hypertext Transfer Protocol - HTTP/1.1", *http://www.ietf.org/rfc/rfc2616.txt*
- RFC2396, "Uniform Resource Identifiers (URI): Generic Syntax", *http://www.ietf.org/rfc/rfc2396.txt*
- RFC1738, "Uniform Resource Locators (URL)", *http://www.ietf.org/rfc/rfc1738.txt*
- C. D. Manning, P. Raghavan and Hinrich Schütze, "Introduction to Information Retrieval", Cambridge University Press. 2008. (http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html)