

la nascita di HTTP

Ing. Cesare Monti 28 febbraio 2005

Cosa vedremo

- HTTP
 - cenni storici
 - differenti modalità di organizzazione della conoscenza
 - nascita di HTTP
 - specifiche
 - limiti
 - sistemi distribuiti
 - client-side
 - server-side

organizzazione della conoscenza

- tramandare conoscenza è esigenza umana
 - forme orali
 - leggende
 - ... pensate ad Omero e alla sua Odissea
 - forme scritte
 - manoscritti
 - biblioteche
- l'uomo lo fa da secoli

organizzazione della conoscenza

- una forma per la rappresentazione come standard “de facto”
 - scrittura lineare
 - usata da millenni
 - deve esserci un filo logico che lega il discorso
 - legata al supporto fisico
 - rappresenta bene le forme organizzate del pensiero umano
 - ma questo non sempre è organizzato

organizzazione della conoscenza

- ...e qualora volessimo rappresentare a pieno il pensiero umano ?
 - teoria dell'ipertesto
 - slegare la rappresentazione dal supporto fisico
 - catturare i nessi e non la sola informazione
 - focalizzarsi sull'informazione come oggetto

organizzazione della conoscenza

- tappe storiche:
 - 1588 - "Le diverse et artificiose macchine del Capitano Agostino Ramelli"
 - 1945 - Vannevar Bush - consigliere di Roosevelt
 - As we may think - Atlantic Monthly
 - Memex - sistema basato sui microfilm
 - 1965 - Theodor Holm Nelson
 - coniò il termine ipertesto
 - Xanadu - dal luogo magico di "Kubla Khan" di S. Taylor

organizzazione della conoscenza

- 1990 - Tim Berners-Lee
 - World Wide Web
 - HTML
 - URLs
 - HTTP
- 1991/92
 - NCSA - Mosaic - primo browser

HTTP

- insieme di RPC (Remote Procedure Calls) basate su TCP/IP
- repository di risorse identificate da URLs
- ogni risorsa è un ipertesto
 - almeno inizialmente
- ad ogni richiesta corrisponde una nuova connessione

HTTP

- Ogni richiesta si compone di due fasi:
 - Request
 - ... fatta da un qualcuno detto Client
 - Response
 - ... fatta da qualcuno detto Server
- ogni fase ha un suo protocollo

HTTP: Request

- Request line (nome del comando da invocare, es: GET)
- Request header field (informazioni aggiuntive, es: parametri di RPC)
- Entity body (riservato al passaggio di informazioni "bulk" al server)

HTTP:Request

- Sintassi:

```
<method><resource identifier><http version> <clrf> } Request line  
[<Header> : <value>] <clrf> }  
: : Request  
: : Header  
[<Header> : <value>] <clrf> } Fields  
blank line <clrf>  
[Entity body] } Entity Body
```

- Esempio:

```
GET /path/to/file.html HTTP/1.0 } Request line  
Accept: text/html }  
Accept: audio/x }  
Accept: image/gif } Request  
User-agent: MacWeb } Header  
 } Fields
```

HTTP: Request

- di Method / Comandi ne esistono diversi !
 - get
 - richiesta semplice
 - head
 - = get ma restituisce solo le intestazioni
 - post
 - chiede di accettare i dati in stream
 - put
 - chiede di memorizzare i dati sulla risorsa URI specificata
 - delete
 - chiede di cancellare la risorsa URI specificata
 - options
 - chiede quali methods siano supportati dal server web
 - trace
 - chiede di mostrare la request http e le sue intestazioni
 - connect
 - ...mai implementato per l'utilizzo di proxy per operazioni di tunneling

HTTP : Response

- Response Header Line (protocollo e numero di errore)
- Response Header Field (informazioni aggiuntive, contenuto, lunghezza ecc...)
- Entity body (il corpo della pagina richiesta)

HTTP: Response

- Sintassi:

```
<HTTP Version> <result code> [<explanation>] <clrf> } Response line  
[<Header> : <value>] <clrf> }  
: : } Response  
: : } Header  
[<Header> : <value>] <clrf> } Fields  
blank line <clrf>
```

[Entity body]

- Esempio:

```
HTTP/1.0 200 OK } Response line  
SERVER: NCSA/1.3 }  
Mime_Version: 1.0 } Response  
Content_type: text/html } Header  
Content_length: 2000 } Fields  
<HTML> }  
: } Entity Body  
: }  
</HTML>
```

HTTP: limiti

- tra ogni richiesta non c'è interazione
- completa perdita di stato ad ogni connessione
 - stateless
 - questo deriva dalla sua progettazione
- ogni RPC è slegata dalla successiva
 - non c'è modo di mantenere la storia dell'interazione
 - decade il legame spazio-tempo

HTTP

Sistemi Distribuiti

- Per ovviare al problema negli anni sono stati sviluppati differenti modelli di interazione:
 - spostare capacità di calcolo sul server
 - sistemi a mainframe
 - spostare capacità di calcolo sul client
 - reti di pc
 - Modelli ibridi
 - architetture client - server

HTTP

Sistemi Distribuiti

- il modello Client - Server oggi è il più diffuso
 - c'è computazione sul client
 - ... e sul server
- nel mondo WEB si traduce in tecnologie abilitanti:
 - client
 - javascript
 - ECMAScript
 - server
 - CGI
 - Application Server

CGI

HTTP: CGI



- Nate per aggiungere interazione tra client e server
- Il codice risiede interamente sulla macchina server
 - ... questo spiega il termine server side ...
- Si lascia aperta la possibilità di eseguire codice chiamandolo dal web
- Questa possibilità aderisce ad un protocollo di Common Gateway Interface

CGI : il protocollo

- I clients richiedono l'esecuzione di un programma
 - ... oggi non sembra nulla... ma allora ...
- I server invocano il programma chiamato nell'URL
- utilizzano il protocollo CGI per interpretare il metodo (GET, POST) con cui passare i parametri al programma invocato (via stdin)
- Il programma viene eseguito e ritorna la risposta in formato HTML (via stdout) al server Web
 - o meglio il programmatore ritorna in HTML via stout...
- Il server Web rigira la risposta al client
- ... e lo stato ???

CGI : Ulteriori migliorie – lo stato

- Tramite CGI è stato introdotto il concetto di stato legato all'HTTP
- Lo stato viene mantenuto attraverso l'invio di variabili che vengono mantenute in memoria dal client (cookies)
 - ... il nome pare derivi da un programmatore Netscape

• <http://www.cookiecentral.com/faq/#1.2>

CGI: how to

- per aggiungere passaggio di parametri tra una connessione e l'altra anche HTML è stato modificato
 - aggiunta di
 - Form
 - `<form action="..." method="...">`
 - Form Elements
 - `<input type="...">`
 - <http://www.w3c.org>
 - <http://www.w3c.org/Consortium/membership>

CGI: How To

- Possono essere scritte in qualsiasi linguaggio che possa venire interpretato dalla macchina server, quelli più usati sono:
 - C/C++
 - Fortran
 - PERL
 - Phyton
 - TCL
 - Any Unix shell
 - Visual Basic
 - AppleScript
 - Java

CGI: il passaggio dei dati

- **GET**
 - viene riscritto run time l'URL della risorsa a cui si vuole accedere aggiungendo i campi che si vogliono passare all'URL stesso
- Es: `<chiamata a: http://www.unSito.com/unoScriptCgi.exe>` con parametri: `param1="10"` , `param2="ciao"`
- Il request line diventa:
 - `GET www.unSito.com/unoScriptCgi.exe?param1="10"¶m2="ciao" HTTP/1.0`

CGI: il passaggio dei dati

- **POST**
 - Tutti i parametri vengono passati dentro al campo Entity Body e viene modificato il method del Request Line
- Es: `<chiamata a: http://www.unSito.com/unoScriptCgi.exe>` con parametri: param1="10" , param2="ciao"
-
- `POST www.unSito.com/unoScriptCgi.exe HTTP/1.0`
...
`Param1="10"`
`¶m2="ciao"`

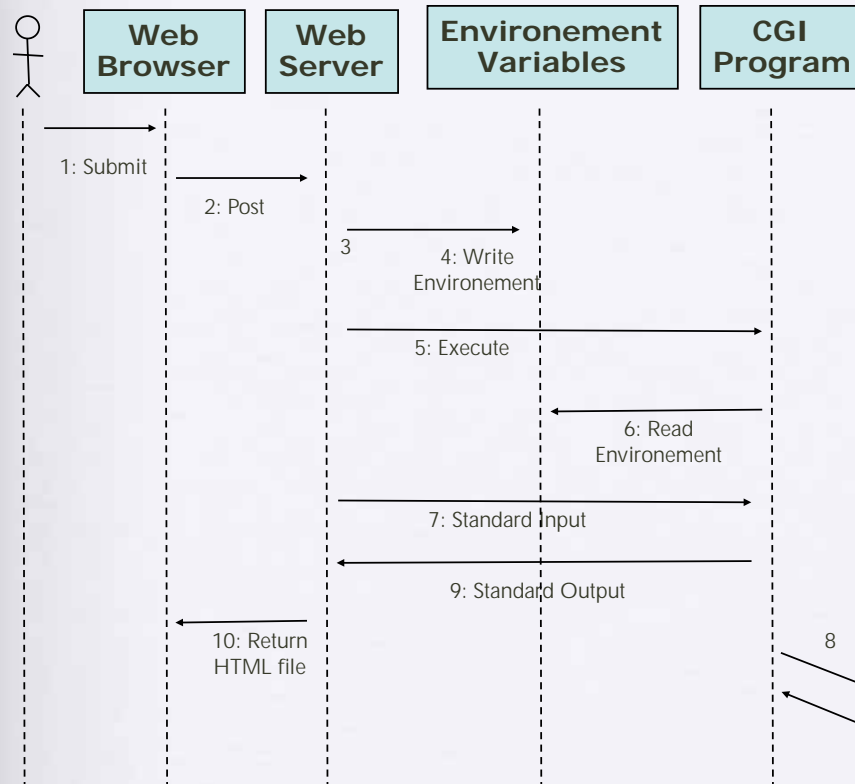
CGI: pro e contro dei metodi

- **GET**
 - **Pro:**
 - i parametri rimangono visibili all'utente
 - l'output della pagina dinamica può essere bookmarcato
 - **Contro:**
 - si deve essere sicuri che lo script CGI reso disponibile non possa eseguire azioni dannose a fronte di parametri sbagliati
 - nella stesura dello script occorre fare parsing sull'URL per avere i parametri

CGI: pro e contro dei metodi

- **POST**
 - **Pro:**
 - non occorre dividere URL dai parametri
 - **Contro:**
 - nel caso di pacchetti incompleti non si può eseguire nulla
 - le pagine non possono essere bookmarcate

CGI: scenario



1. L'Utente manifesta la volontà di sottomettere dati ad elaborazione, il browser colleziona i dati, controlla il metodo da invocare e genera la request

2. Il browser invoca il metodo POST e sottomette la request

3. Il Server riceve la richiesta e inizia la sessione di risposta

4. Il Server setta le variabili di environment (server name, request method, path_info, content_type ...)

5. Il server HTTP da inizio al programma CGI

6. Il programma CGI legge le variabili d'ambiente

7. Riceve da stdin i dati utente

8. Fa qualcosa

9. Ritorna i risultati su stdout

10. Il server HTTP formatta il risultato e lo rispedisce al client

Example: C code

```
main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int cl;
    printf("Content-type: text/html%c%c",10,10);
    // CHECK SUL CONTENT TYPE
    if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-urlencoded")) {
        printf("This script can only be used to decode form results. \n");
        exit(1);
    }
    cl = atoi(getenv("CONTENT_LENGTH"));
    // AQUISIZIONE DATI DA STDIN
    for(x=0;cl && (!feof(stdin));x++) {
        m=x;
        entries[x].val = fmakeword(stdin,'&",&cl);
        entries[x].name = makeword(entries[x].val,'=');
    }
    // PREPARAZIONE DATI DI OUTPUT
    printf("<H1>Query Results</H1>");
    printf("You submitted the following name/value pairs:<p>%c",10);
    printf("<ul>%c",10);

    for(x=0; x <= m; x++)
        printf("<li> <code>%s = %s</code>%c",entries[x].name,
            entries[x].val,10);
    printf("</ul>%c",10);
}
```

CGI: dove sta la fregatura?

- Il protocollo CGI prevede l'istanziamento di un nuovo processo ogni qual volta si invochi una CGI
 - pensate quindi che ad ogni request parte un processo
 - pensate ad un server web con molta utenza ...
 - col tempo è stata introdotto il protocollo FastCGI ... ma non sempre è applicabile

CGI e dopo?

- scrivere CGI implica scrivere un applicativo che produca il proprio output codificato in HTML
- implica anche una serie impressionante di problemi di gestione
 - legati alla scalabilità dell'ambiente
 - ed alla eterogeneità dei client
 - quanti applicativi Browser esistono??

CGI e dopo ???

- col tempo l'evoluzione delle CGI ha portato ad una serie di:
 - linguaggi di elaborazione server-side
 - PHP
 - JSP
 - ASP
 - Ognuno dei quali ha propri meccanismi per abilitare e gestire l'interazione e la comunicazione
 - beans
 - sessioni
 - oggetti server

CGI e dopo ??

- evoluzione degli applicativi server
 - da web server a Application server
 - JBoss
 - Cocoon
 - ...
 - ognuno con proprie specifiche e capacità

CGI

- un passo per volta
 - qualche link
 - <http://www.cgi101.com/book/>
 - <http://cgipoint.html.it/tutorial/>
 - <http://hjs.geol.uib.no/Cplusplus/>
 - un esempio per tutti che guarderemo insieme
 - <http://www.sampublishing.com/articles/article.asp?p=101661&seqNum=1>