

Elaborazioni server-side: dalle CGI a JSP

Ing. Cesare Monti 5 maggio 2004

1

Cosa vedremo

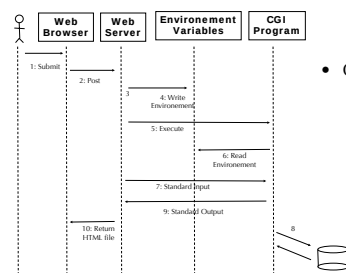
- HTTP
- Cosa sono le CGI
- Cos'è JSP:
 - il contesto:
 - in cui c'è bisogno di JSP
 - di cui ha bisogno JSP
 - a byte of syntax

2

HTTP: from stateless to interaction

3

Dove siamo



- Cercheremo di interpretarlo

4

HTTP

HTTP: Web's RPC (Remote Procedure Calls) basate su TCP/IP

Il web viene visto come un grande URL-based file server

Il web su http è centrato sul concetto di Documento e di ipertesto

HTTP è senza stato (stateless)

Ogni richiesta corrisponde ad una nuova connessione

Non c'è interazione tra una richiesta e l'altra

5

Il protocollo HTTP

- Si basa su due fasi:
 - Richiesta (HTTP:Request)
 - Risposta (HTTP:Response)
- Ognuna di queste fasi ha un suo possibile protocollo:
 - Request: {line, header, body}
 - Response: {header line, headers fields, body}

6

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)

7

HTTP:Request

```

- Sintassi:
<method><resource identifier><http version> <clrf>      Request line
[<Header> : <value>] <clrf>                               Request
:                                                         Header
:                                                         Fields
[<Header> : <value>] <clrf>
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
User-agent: MacWeb                    Request
                                     Header
                                     Fields
    
```

8

HTTP: Request

- di Method 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

9

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)

10

HTTP: Response

```

- Sintassi:
<HTTP Version> <result code> [<explanation>] <clrf>      Response line
[<Header> : <value>] <clrf>                               Response
:                                                         Header
:                                                         Fields
[<Header> : <value>] <clrf>
blank line <clrf>
[Entity body]                                             Entity Body

- Esempio:
HTTP/1.0 200 OK      Response line
SERVER: NCSA/1.3
Mime_Version: 1.0
Content_type: text/html
Content_length: 2000
<HTML>
:
:
</HTML>
    
```

11

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

12

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

13

CGI : Ulteriori miglione – 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>

14

CGI: How To

- Possono essere scritte in qualsiasi linguaggio che possa venire interpretato dalla macchina server, quelli più usati sono:

- C/C++
- Fortran
- PERL
- Python
- TCL
- Any Unix shell
- Visual Basic
- AppleScript
- Java

15

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

16

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"

17

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

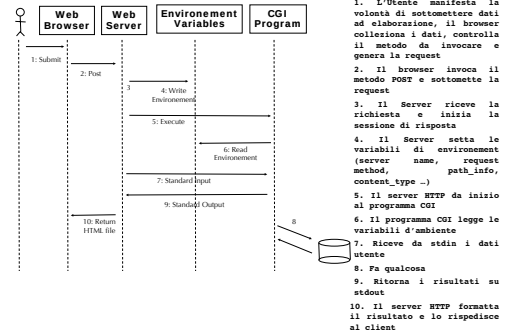
18

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 bookmarkate

19

CGI: scenario



20

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

21

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 ...
 - ...è sì vero che il FastCGI ha risolto il problema ma rimane la difficoltà di gestire il tutto

22

JSP

23

Java Server Pages

- JSP means Java Server Pages
 - ...il nome è tutto un programma
- nasce per il bisogno di server-side
 - nel '97 viene pubblicata una API chiamata Java Servlet
 - si occupava di servlet ... qualunque cosa siano ...
 - nel '98 viene modificata in API JSP
 - si occupava (e lo fa ancora) di interazione fra WEB e Servlet

24

JSP: 97 Vs 98

- 97 - Servlet API
 - potenza di elaborazione tra classi java che usano HTML come meccanismo per standardizzare l'output
- 98 - JSP API
 - pagine html con Java "incastonato"

25

JSP: il contesto

- siamo sul web, hic sunt:
 - server web
 - DB
 - beans, bundles,
 - ... oggetti in generale ...
 - agenti ... tra non molto

26

JSP: il contesto

- all'epoca la prima sfida fu quella di aggiungere computazione lato client
- Java fece le Applet ... con i relativi problemi di:
 - esecuzione
 - affidabilità
 - sicurezza
 - Microsoft Java Machine (quella di allora!)
- all'epoca tempi lunghi per spostare codice
 - non c'era la banda larga
- poche applicazioni grafiche per contenere le dimensioni

27

JSP: il contesto

- si cercavano quindi "maniere per uniformare"
 - e si tornò al server side
- prima
 - con maniere standard di far scrivere output (html) da classi java (servlet)
- poi
 - con nuove maniere di scrivere "web",
 - java dentro html

28

JSP: il contesto

- a bit of history: Model Vs Model
 - con le servlet la sfida fu di abilitare "automi software" per la scrittura di html
- con jsp la sfida cambiò focus, si cercava di scrivere codice html (tag) a cui fosse associato codice java da eseguire.
 - di quale sia l'impatto di questo cambio di focus ne discuteremo più avanti

29

JSP: il contesto

- per riuscire nel tutto occorre qualche anno
- si realizzarono i "container" conformi alle API JSP
 - Apache Jakarta Tomcat
 - Resin servlet / JSP container
 - WebSphere Application Server
 - GNUJSP
 - Jetty

30

JSP: il contesto

- ... e le stesse specifiche JSP cambiarono:
 - dalla API 1.0 ...alle 2.0
 - ... ora le 2.2
 - con tanti pro e qualche contro

31

JSP: il contesto

- volendo fissare il contesto:
 - JSP opera server-side
 - associa codice java a tag HTML
 - interagisce con java in ogni sua forma
 - ha bisogno di un ambiente consono alla sua vita

32

JSP: il contesto

- le pagine JSP abbisognano di un contenitore adatto
 - perchè ?
 - ... se c'è del codice qualcuno lo deve compilare e visto che è java pure interpretarlo
 - chi lo fa?
 - (retorica) il container o engine che dir si voglia

33

JSP: il contesto

- guardiamo il container:
 - deve stare sul web,
 - rispondere a richieste http (sicuramente un server web)
 - compilare e interpretare java (sicuramente avrà un JVM)
 - un esempio per tutti:
 - Jakarta Tomcat

34

JSP: il contesto

- Tomcat
 - nasce come progetto Open Source
 - nasce come una falange di apache ... poi diviene un server web a se stante
 - <http://jakarta.apache.org>
 - ... è alla sua quinta release (2004)
 - è più lento di apache nella risposta http normale
 - ... ma risponde anche all'elaborazione di jsp e pare sia meglio degli altri

35

JSP: il contesto

- Tomcat
 - prevede una struttura pubblica delle pagine organizzate in directory
 - come apache
 - compila le pagine alla prima loro invocazione
 - le interpreta ogni volta che vengono chiamate

36

JSP: la sintassi

- si è detto java dentro html ergo ci sono dei tag di marcatura
- `<%` (apro il tag)
- `%>` (chiudo il tag)
 - `<%-- questo è un esempio di un commento --%>`
- ... e non solo, ho tag appartenenti ad uno specifico namespace
 - `<jsp:forward page="URL" />`

37

JSP: la sintassi

```
<html>
<head><title>Prima pagina JSP </title></head>
<body><table><tr>
  <% for (int i= 0; i<=3; i++){
    %> <td>
      <%= "&nbsp;" + java.util.date() + "&nbsp;" %>
    </td> <%
  } %>
</tr></table></body></html>
```

38

JSP: la sintassi

- l'esempio di prima fa pensare...
 - uso un package ... di solito in java li importo ... cosa devo fare qui?
 - lo stesso
- esiste una direttiva di import dei packages
 - che nell'esempio di prima non c'era!
 - `<%@ page import="java.util.*" %>`
 - ha validità solo per la pagina così come in java ha validità per la classe in cui la si usa
 - dicansi ... direttive di pagina

39

JSP: la sintassi

- Le direttive di pagina sono ciò che ci permettono di specificare il dominio e l'ontologia della pagina jsp
- `<%@ page language="italian"`
- `buffer="dimensione del buffer dell'output"`
- `info="descrizione di pagina"`
- `errorPage="URL pagina di risposta in errore"`
- `contentType="tipo Multipurpose Internet Mail Extension" %>`
- `taglib uri="URI"`

40

JSP: la sintassi

- ... per la modularità possiamo includere file esterni
- a compile-time
 - `<%@ include file="URL"%>`
 - ... che è ancora una direttiva di pagina
- a request-time
 - `<jsp:include page="URL" flush="true"/>`
 - ...che non è più una direttiva di pagina ma un tag a tutti gli effetti,
 - posso specificare anche i parametri da passare alla pagina (per coerenza con il protocollo CGI)
 - `<jsp:include page="URL" flush="true">`
 - `<jsp:param name="first" value="Hello" />`
 - `<jsp:param name="second" value="World" />`
 - `</jsp:include>`

41

JSP: la sintassi

- ... e le dichiarazioni di carattere globale
 - `<%! String stringa="Hello World" %>`
- questa viene vista in tutto ciò che viene a "contatto" con la pagina in cui è inserita
 - sia le possibili classi importate a priori
 - sia quello incluse a posteriori

42

JSP: la sintassi

- la slide precedente è valida anche per le funzioni

```
<%! int somma(int a, int b){  
    return a+b;  
} %>
```

- e anche in questo caso la funzione è vista da tutto ciò che viene a contatto con la pagina

43

JSP: la sintassi

```
...  
<form name="form1" method="GET | POST" action="Azione.jsp">  
<input type="text" name="user">  
<input type="password" name="pw">  
<input type="submit" name="invia dati">  
</form>  
...
```

44

JSP: la sintassi

```
<%@ taglib uri="WEB-INF/tlds/html.tld" prefix="html"%>
```

...

```
<td> <html:requestParameter property="user"></td>
```

```
<td><html:requestParameter property="pw"></td>
```

...

- e qui ho già recuperato i valori passati dalle form ... operazione che in CGI avrebbe richiesto molta più fatica
- ho inoltre scoperto che esistono tag o librerie di tag per manipolare l'output
 - quali sono tutti ? ... questi li trovate in rete ...
 - e se volessi farne dei miei? ... si possono creare, esistono le specifiche dei TAG standard (sempre in crescita) ma nulla vieta di crearsene dei propri (java.servlet.jsp.tagext)

45