Basi di dati & Web

[Parzialmente tratte da "Tecnologie Web e Database"
ing. Marco De Paola -IT Professional
Corso di Basi di dati e Sistemi Informativi (Diploma),
Università degli Studi di Milano, a.a. 2000/2001,
http://www.disi.unige.it/person/MerloI/teach.html]

1

Sommario

- Il World Wide Web (WWW)
  - L'architettura Client-Server del WWW
  - Standard utilizzati nel WWW
  - Interazione tra Browser e Web Server
  - Gli script, i gateway e le form del Web
- Tecnologie per l'integrazione Web/Basi di dati
  - introduzione
  - tecnologie server side
  - tecnologie client side

2

Il World Wide Web

3

Il World Wide Web (WWW)

Il World Wide Web (detto anche Web, WWW o W3) è nato al Cern nel 1989 per consentire una agevole cooperazione fra i gruppi di ricerca di fisica sparsi nel mondo

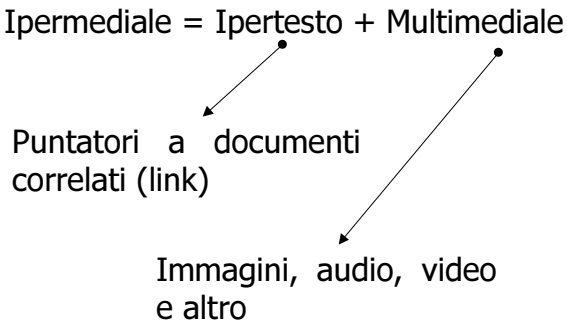
Che cos'è il WWW

"*E' un'architettura software volta a fornire l'accesso e la navigazione a un enorme insieme di documenti collegati fra loro e sparsi su milioni di elaboratori*"

4

Il World Wide Web

Tali documenti si dicono *ipermediali* che è la combinazione delle parole ipertesto e multimediale, infatti essi contengono puntatori ad altri documenti ed informazioni multimediali come immagini, audio, video ecc.



5

Il World Wide Web

Il Web ha diverse caratteristiche che hanno contribuito al suo enorme successo:

1) Architettura di tipo client-server:
   - ampia scalabilità, adatta ad ambienti di rete;

2) Architettura distribuita:
   - perfettamente in linea con le esigenze di gestione di un ipertesto;

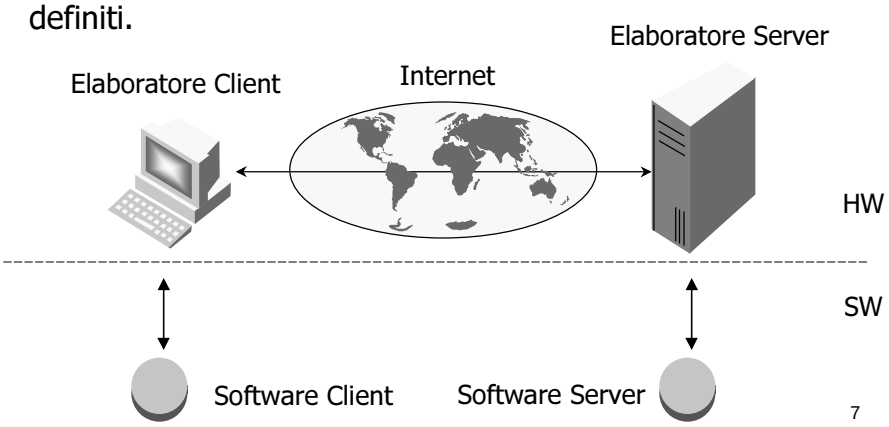
3) Architettura basata su standard di pubblico dominio:
   - uguali possibilità di accesso per tutte le piattaforme di calcolo;

4) Capacità di gestire informazioni eterogenee :
   - testo, immagini, suoni, filmati, realtà virtuale, ecc.

6

L'architettura client-server del WWW

Il Web è una architettura software di tipo *client-server*, nella quale sono previste due tipologie di componenti software: il *client* e il *server*, ciascuno avente compiti ben definiti.



7

L'architettura client-server del WWW

Il *client* è lo strumento a disposizione dell'utente che gli permette l'accesso e la navigazione nell'ipertesto del Web. Esso ha varie competenze tra le quali:

1) Trasmettere all'opportuno server le richieste di reperimento dati che derivano dalle azioni dell'utente

2) Ricevere dal server le informazioni richieste

3) Visualizzare il contenuto della pagina Web richiesta dall'utente, gestendo tutte le tipologie di informazioni in esse contenute

I client vengono comunemente chiamati *browser* (sfogliatori) (es: Netscape Navigator; Internet Explorer).

8

L'architettura client-server del WWW

Il *server* è tipicamente un processo in esecuzione su un elaboratore. Esso è sempre in esecuzione ed ha delle incombenze molto semplici:

1) rimanere in ascolto di richieste da parte dei client

2) fare del suo meglio per soddisfare ogni richiesta che arriva:

- se possibile, consegnare il documento richiesto

- altrimenti, spedire un messaggio di notifica di errore (documento non esistente, documento protetto, ecc.)

I server vengono comunemente chiamati *demoni-HTTP* o *web server*

9

Standard utilizzati nel WWW

Ci sono tre standard principali che, nel loro insieme, costituiscono l'architettura software del Web:

1) sistema di indirizzamento basato su *Uniform Resource Locator (URL):* è un meccanismo standard per fare riferimento alle entità indirizzabili (risorse) nel Web

2) *HTML (HyperText Markup Language):* è il linguaggio per la definizione delle pagine Web

3) protocollo *HTTP (HyperText Transfer Protocol):* è il protocollo che i client e i server utilizzano per comunicare

10

Uniform Resource Locator (URL)

Una URL costituisce un riferimento a una qualunque risorsa accessibile nel Web.

Tale risorsa ovviamente risiede da qualche parte, dunque, una URL deve essere in grado di indicare:

1) come si vuole accedere alla risorsa

2) dove è fisicamente localizzata la risorsa

3) come è identificata la risorsa

Per queste ragioni, una URL è fatta di 3 parti, che specificano:

metodo di accesso://Nome host/identità risorsa

http://somewhere.net/pagina.html

11

Hypertext Markup Language (HTML)

Il linguaggio per la formattazione di testo HTML è una specializzazione del linguaggio SGML (Standard Generalized Markup Language) definito nello standard ISO 8879.

Un markup language si chiama così perché i comandi (tag) per la formattazione sono inseriti in modo esplicito nel testo.

ES : <B>Hello World !!!</B>

Visualizzazione nel browser : **Hello World !!!**

12

Hypertext Transfer Protocol (HTTP)

Il protocollo HTTP sovraintende al dialogo fra un client e un server web, ed è il linguaggio nativo del Web.

HTTP è un protocollo ASCII, cioè i messaggi scambiati fra client e server sono costituiti da sequenze di caratteri.

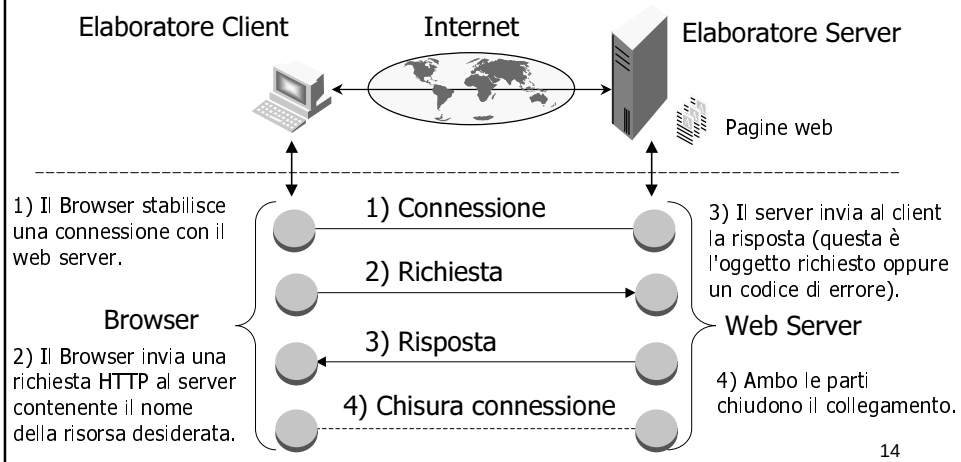
Il protocollo è di tipo *stateless*, cioè non è previsto il concetto di sessione all'interno della quale ci si ricorda dello stato dell'interazione fra client e server. Ogni singola interazione è storia a sé ed è del tutto indipendente dalle altre.

Il fatto di essere senza stato è uno dei punti critici nelle *transazioni commerciali*.

13

Interazione tra browser e Web server

L'interazione tra il browser (client) ed il Web server avviene in 4 fasi :



14

Gli Script, i Gateway e le form del Web

Il WWW è utilizzato fondamentalmente per pubblicare e distribuire le informazioni agli utenti.

Questi servizi sono essenzialmente a senso unico *(broadcast)*

L'utilizzo del web da parte di molte organizzazioni commerciali ha portato ad una richiesta di traffico a due sensi, quindi ad una interazione tra utente e server HTTP.

15

Gli Script, i Gateway e le form del Web

Il traffico a due sensi è realizzabile utilizzando programmi ausiliari che vengono eseguiti sulla piattaforma del server. Tali programmi vengono chiamati *script o gateway.*

L'idea fondamentale che sta alla base degli script del web è che alcuni documenti non sono in realtà dei veri documenti ma dei veri e propri programmi eseguibili o interpretati scritti in un qualsiasi linguaggio di programmazione come *C, Basic o Perl*

Gli script sono in grado di raccogliere informazioni da parte dell'utente mediante particolari pagine HTML interattive chiamate *form*.

16

Web & basi di dati

- Obiettivi: :
  - ottenere la generazione dinamica di pagine a partire da dati contenuti in una base di dati
  - sfruttare i pregi di Web e basi di dati, aggirandone i difetti

17

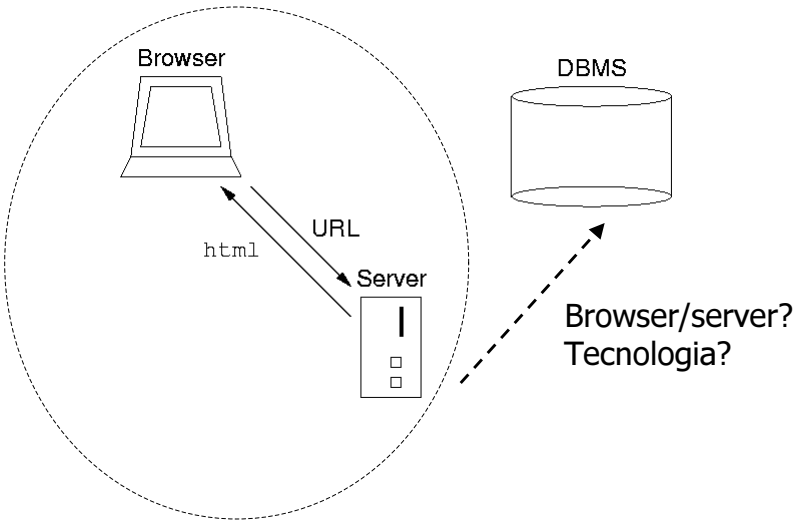
Pregi e difetti di basi di dati e Web

- **Web**
- Pro
  - semplice
  - portabile
  - a basso costo
  - indipendente dalle interfacce
  - ipermediale
- Contro
  - basato su file
  - statico

- **Basi di dati**
- Pro
  - modelli dei dati
  - linguaggi di interrogazione
  - funzioni di amministrazione
- Contro
  - complesse
  - proprietarie
  - navigazione e presentazone assenti

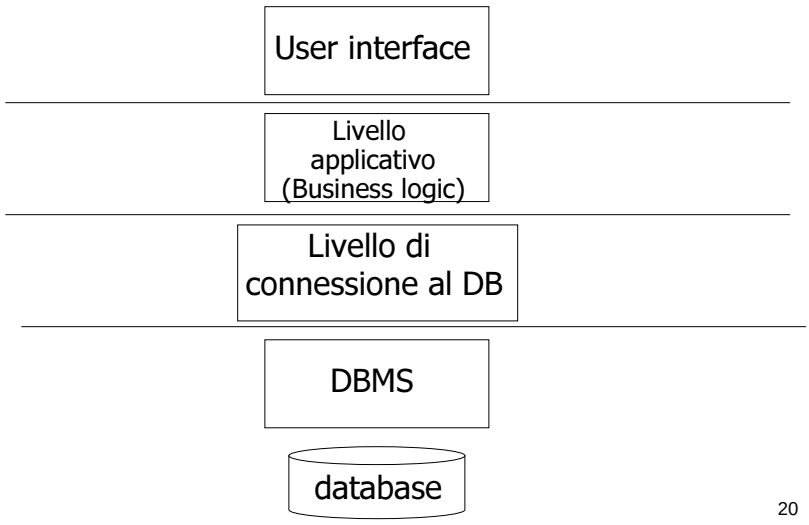
18

Problema



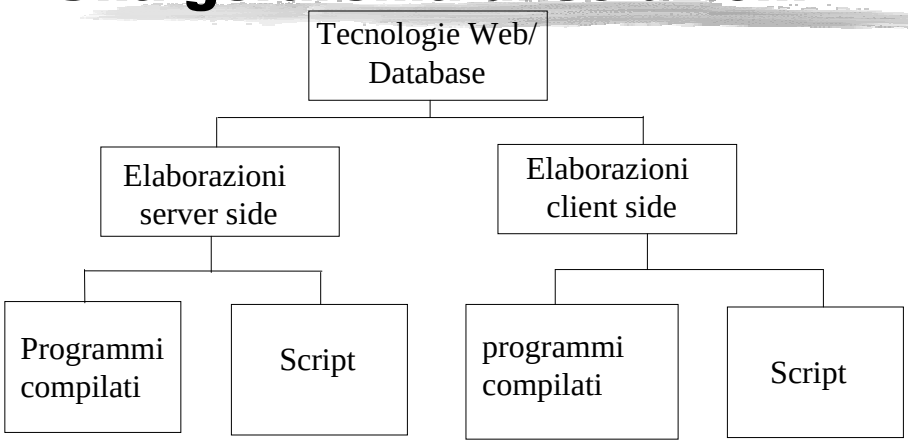
19

Architettura generale a livelli



20

Una gerarchia di soluzioni



Programmi per servizi CGI
Java Servlet

ASP
PHP
JSP

Java Applet
programmi Active X

JavaScript
VBScript

21

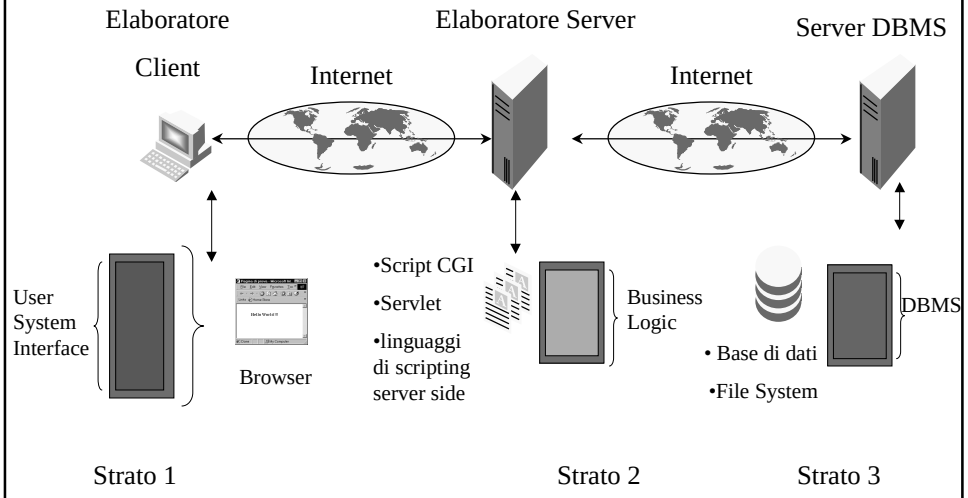
Soluzioni lato server

Vedremo:

- **CGI**
- **Java Servlet**
- **Java Server Pages (cenni)**

22

Sistema classico a 3-strati (lato server)



23

Common Gateway Interface

Quando un browser client richiede un URL di qualcosa che in realtà è un programma il software del server HTTP ha un semplice ruolo: avvia lo script e passa i dati dal browser allo script e viceversa.

Il passaggio corretto dei dati tra il processo demone e lo script è garantito dalla CGI (Common gateway interface) che è uno standard su come devono essere chiamati gli script e come vengono passati i dati.

Dal punto di vista dello script, l'input proviene dal client e l'output ritorna al client; il web server non rientra nel quadro.

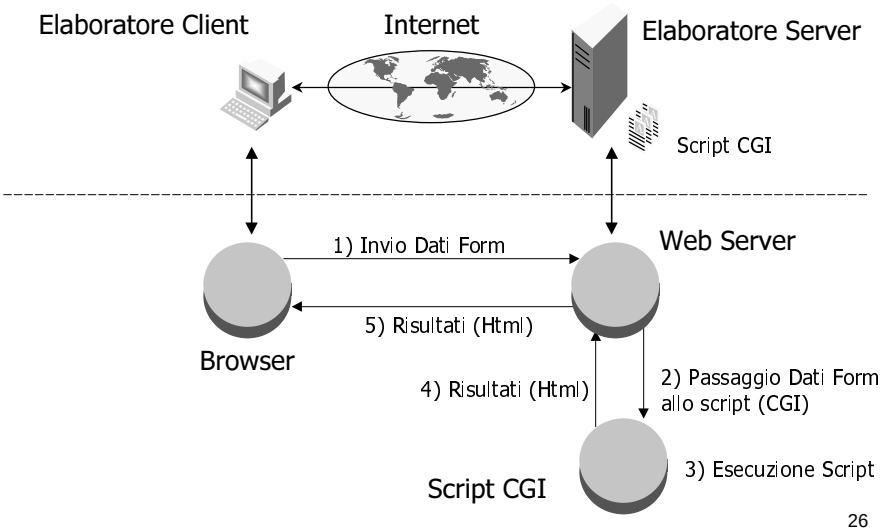
24

CGI

- Quindi CGI è un protocollo che consente al Web Server di eseguire applicazioni esterne in grado di creare pagine dinamicamente
- Definisce un insieme di variabili di ambiente utili alla applicazione (ad esempio, parametri inviati dal client)
- L'applicazione può essere scritta in un qualsiasi linguaggio (C, Java, linguaggi per script) o usare i comandi di una shell
- Gli eseguibili devono essere inseriti in una directory gestita dal Web Administrator (/cgi-bin)
- Il Web Server deve permettere la gestione delle variabili di ambiente

25

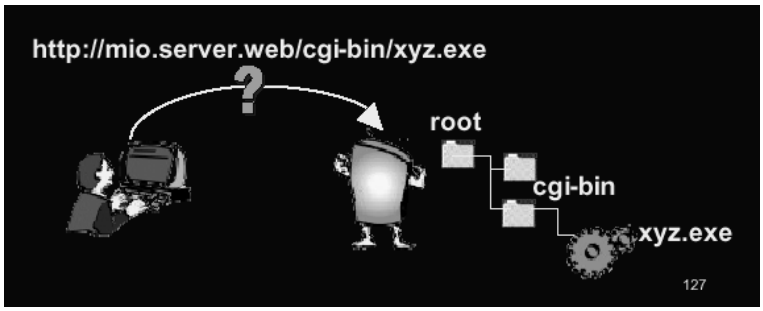
CGI



26

Common Gateway Interface: passo 1

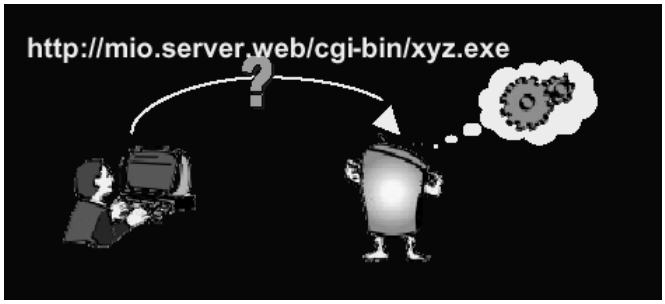
- Il cliente specifica nell'URL il nome del programma da eseguire
- Il programma deve stare in una posizione precisa (di solito il direttorio cgi-bin)



27

Common Gateway Interface: passo 1

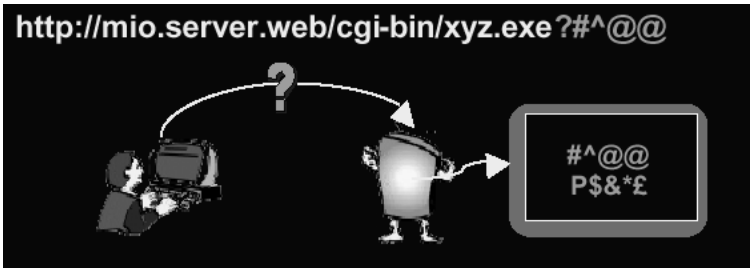
- Il server riconosce dall'URl che la risorsa richiesta dal cliente e' un eseguibile



28

Common Gateway Interface: passo 2

- Il server decodifica i parametri inviati dal cliente e riempie le variabili d'ambiente es: request_method, query_string, content_length, content_type



29

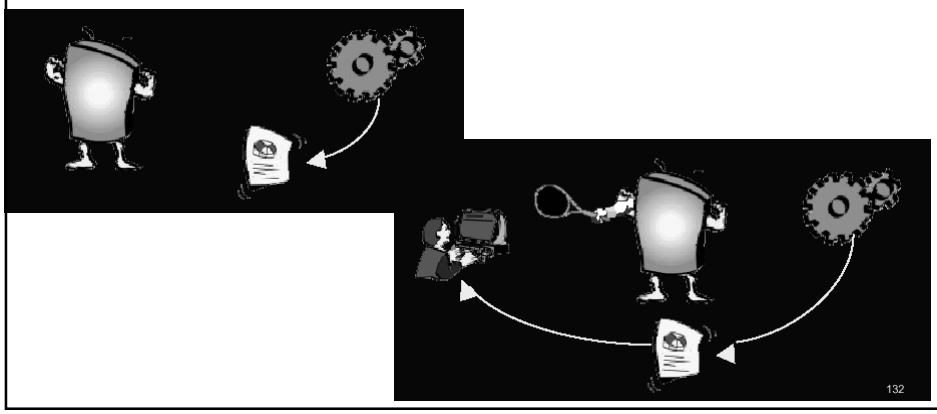
Common Gateway Interface: passo 3

- Il server lancia in esecuzione l'applicazione richiesta
- l'applicazione può utilizzare le variabili d'ambiente

30

Common Gateway Interface: passi 4-5

- L'applicazione stampa la sua risposta (nuova pagina HTML) sullo standard output e il server ridireziona automaticamente lo standard output sulla rete quindi verso il client



132

Invio di parametri ad un programma CGI

- In genere, il client invia informazioni al server tramite FORM HTML
- Le FORM permettono di leggere input da una pagina Web
- tramite il tag ACTION si possono eseguire CGI-bin passando come parametro l'input inviato dall'utente tramite il tag SUBMIT

```
<FORM ACTION="http://servername/cgi-bin/test" METHOD= "GET">
<INPUT TYPE = "TEXT" NAME="Campo"  VALUE="">
<INPUT TYPE = "SUBMIT" NAME="Submit"  VALUE="SEND">
```

- ACTION event handler per l'evento SUBMIT

32

Invio di parametri ad un programma CGI

Due metodi:

- GET: i parametri sono "appesi" all'indirizzo URL specificato con ACTION (recuperabile dalla variabile QUERY_STRING)

```
<FORM ACTION="http://servername/cgi-bin/test" METHOD= "GET">
<INPUT TYPE = "TEXT" NAME="Campo1"  VALUE="">
<INPUT TYPE = "TEXT" NAME="Campo2"  VALUE="">
<INPUT TYPE = "SUBMIT" NAME="Submit"  VALUE="SEND">
```

  - Il server riceve nella variabile QUERY_STRING:

```
Campo1 = Valore1 & Campo2 = Valore2
```

- POST: i dati vengono inviati al server in una linea separata, senza limite al numero di caratteri inviati (recuperabili dallo standard input)

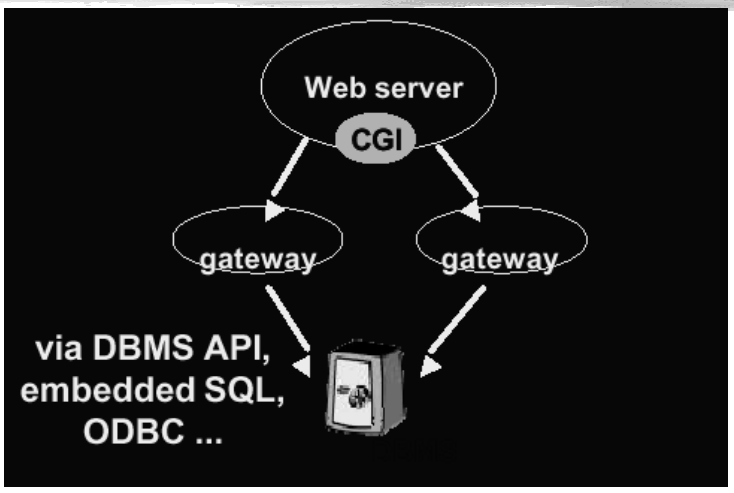
```
<FORM ACTION="http://servername/cgi-bin/test" METHOD= "POST">
<INPUT TYPE = "TEXT" NAME="Campo1"  VALUE="">
<INPUT TYPE = "TEXT" NAME="Campo2"  VALUE="">
<INPUT TYPE = "SUBMIT" NAME="Submit"  VALUE="SEND">
```

  - Il server riceve dallo standard input:

```
http://servername/cgi-bin/test ?Campo1 = Valore1 & Campo2 =
Valore2
```

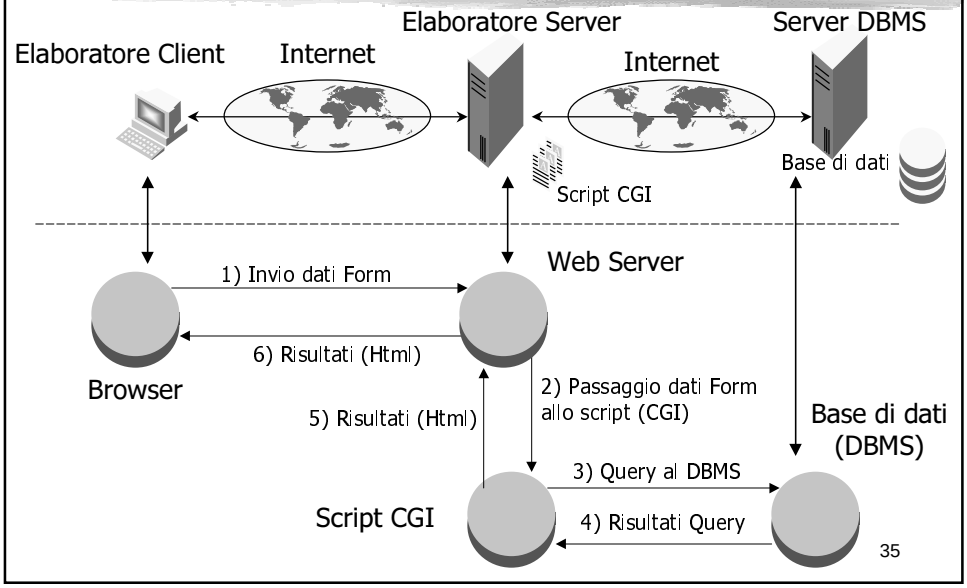
33

CGI e basi di dati



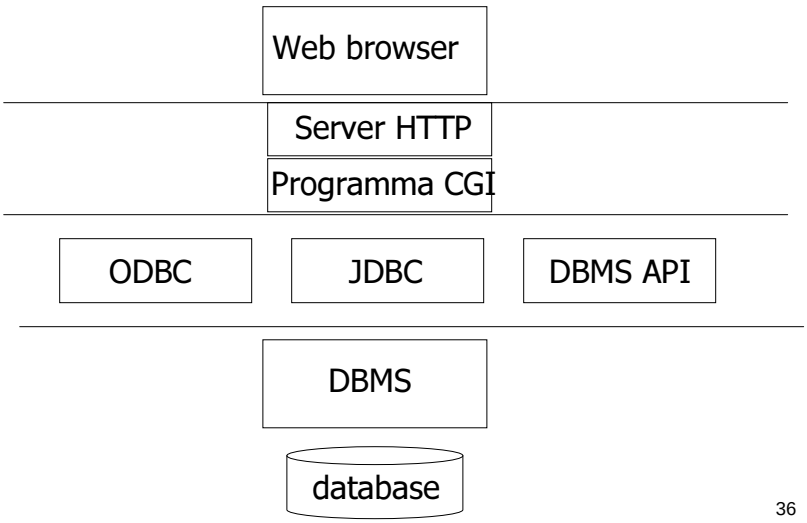
34

CGI : interazione con base di dati



35

CGI: livelli



36

Esempio: Form

```
<FORM ACTION="/cgi-bin/dbconn.cgi" METHOD="POST" >
  LOGIN:
  <INPUT TYPE="text" NAME="login" VALUE=""><p>
  PASSWORD:
  <INPUT SIZE=40 TYPE="password" NAME="password" VALUE="">
  <INPUT TYPE="submit" VALUE="Submit">. <P>
 </FORM>
```


Esempio:CGI per accesso a DB

```
import java.util.*;
import java.io.*;
import java.sql.*;
import java.cgi_lib.*;

class DBconn {
  public static void main( String args[] )
  {
      //  Parserizzo i dati ricevuti sullo standard input
      //  in una hash table indicizzata su NAME //
      Hashtable form_data = cgi_lib.ReadParse(System.in);
      // recupero i dati //
       String my_login = (String)form_data.get("login");
       String my_password = (String)form_data.get("password");
```

Esempio

```
try {
     // definisco il driver JDBC di comunicazione con DB2
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
    // creazione connessione al DB
     String url = "jdbc:db2:H_Temp";
     con = DriverManager.getConnection(url, my_login,
                  my_password);
     // creo pagina Web
     String page = "<html>\n" +
             "<title>Validate Page</title>\n" +
              "Utente registrato\n" +
             "</html> \n";
     // invio pagina al client tramite standard output
     System.out.println(page);
           }
```


Esempio

```
// se viene generata un'eccezione, invio pagina HTML di
   errore
  catch(Exception e) {
     e.printStackTrace();
     String page ="A problem occured while recording" +
     "your answers.\n" + "Please try again.");
      System.out.println(page);
   }
   con.close();
}
```

CGI e basi di dati

- Portabilità:
  - usa solo standard aperti: URL, HTTP, CGI, HTML
  - un cambiamento di piattaforma richiede la ricompilazione del codice sorgente
  - un cambiamento di DBMS richiede di aggiornare gli accessi e ricompilare il codice (a meno che non si usi ODBC, JDBC)
- Prestazioni:
  - ogni accesso richiede:
    - caricare un programma in main memory
    - aprire una connessione con il DBMS
    - eseguire un insieme di query
    - chiudere la connessione
  - quindi: DBMS e SO sono sovraccarichi
- Mantenibilità
  - la presentazione è codificata nel codice sorgente
  - l'accesso ai dati è codificato nel codice sorgente

41

Java servlet

- Java Servlet si riferisce ad un particolare package: `java.servlet`
- con Java Servlet intendiamo un'applicazione Java in esecuzione su una JVM residente sul server
- fornisce una serie di servizi in base al paradigma "richiesta-risposta":
  - client invoca la servlet nel contesto di una FORM HTML (come CGI)
  - la servlet esegue le operazioni richieste (come CGI)
  - la servlet ridirige l'output al client in forma di pagina HTML (come CGI)
- Differenza rispetto a CGI: la servlet corrisponde ad un processo che viene caricato solo una volta anche se viene utilizzato per eseguire più operazioni distinte

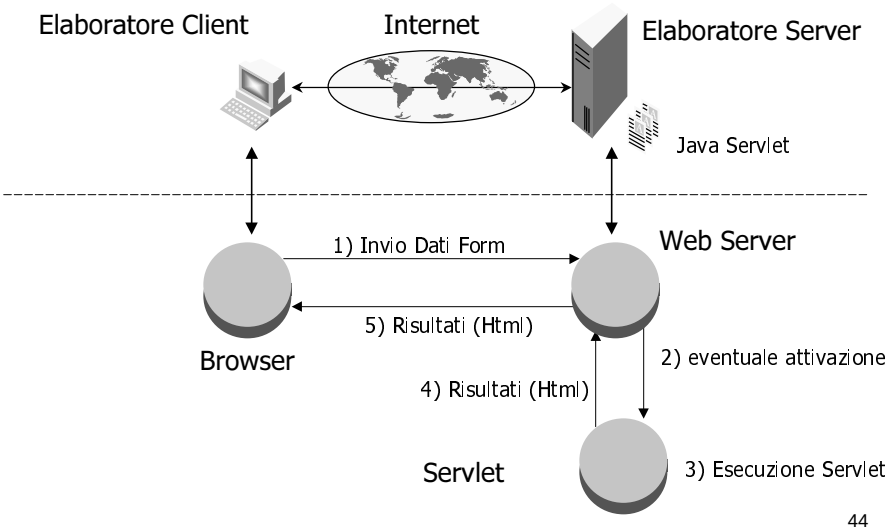
42

Java Servlet

- Altri vantaggi:
  - indipendenza dalla piattaforma, grazie a Java
  - sicurezza gestita mediante security manager della JVM
  - gestione degli errori con il meccanismo delle eccezioni Java
  - disponibilità: distribuzione gratuita di Java Servlet Development Kit, contenente la libreria Java Servlet
- Web Server deve essere esteso con un servlet engine

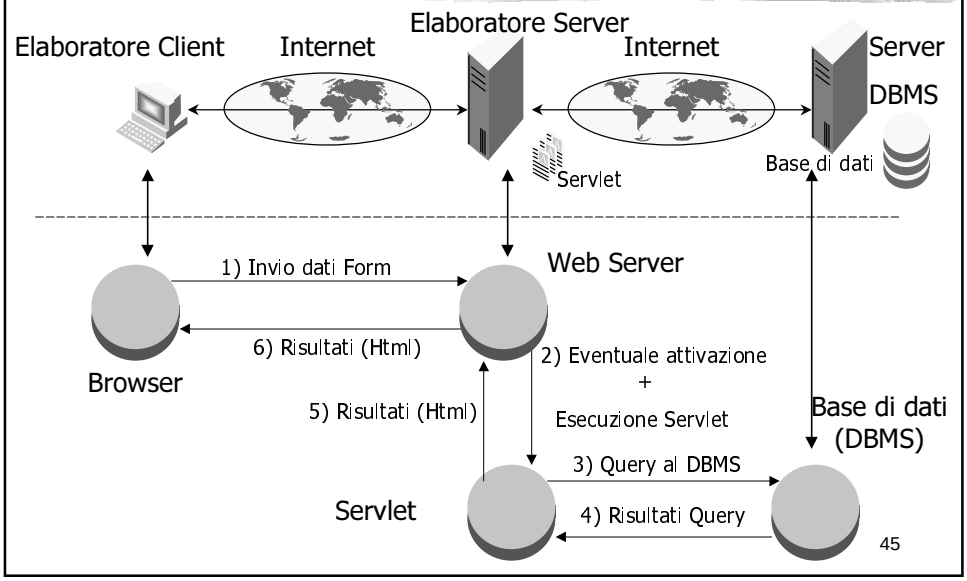
43

Servlet: funzionamento



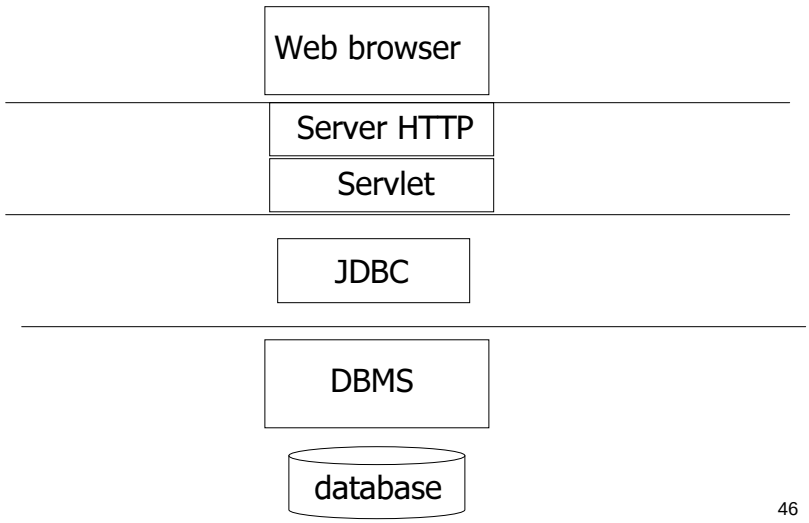
44

Servlet: interazione con base di dati



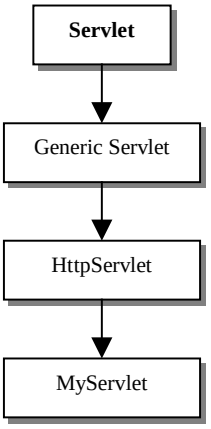
45

Servlet: livelli



46

Architettura del package servlet



Interfaccia Servlet

Classe Generic Servlet

Classe HttpServlet che contiene i metodi per la gestione richieste tramite protocollo HTTP

Classe MyServlet definita dall'utente per soddisfare particolari operazioni

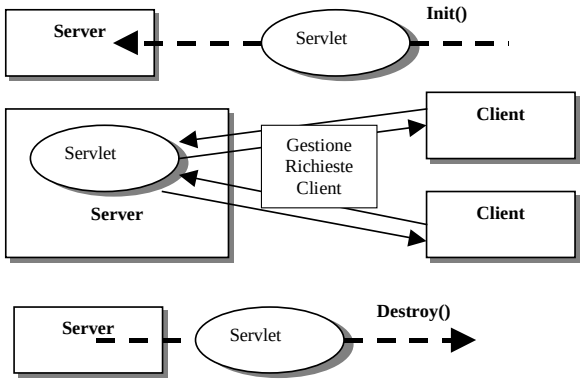
47

Ciclo di vita

Caricamento e inizializzazione

Gestione richieste

Rimozione servlet



48

Esempio di dichiarazione

```
// definizione di una SERVLET per gestire la connessione ad
   un DB
// nome servlet: validate
import java.io.*;
import iava.net.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class validate extends HttpServlet {
// definizione della classe di connessione
Connection con;
// implementazione metodo init( )
// implementazione metodo destroy( )
// implementazione metodo doGet( ) o doPost( ) per soddisfare
// richieste di servizi da parte del client
} // fine class validate
```

49

Caricamento

- La documentazione del Web Server specifica la posizione nella quale inserire l'applicazione servlet
- le servlet vengono caricate dinamicamente dal Web Server alla prima richiesta del client
- vengono eseguite in parallelo all'interno dello stesso processo del server

50

Inizializzazione

- Dopo avere caricato la servlet, la servlet viene inizializzata invocando il metodo `init()`
- Ogni servlet deve ridefinire tale metodo, contenuto nella classe `HttpServlet`
- Tale metodo viene eseguito solo una volta
- Può essere rieseguito dal server solo dopo la rimozione della servlet
- In caso di interazione con il DBMS, tale metodo potrà ragionevolmente realizzare la connessione con il database, che in questo modo verrà realizzata solo una volta

51

Esempio Inizializzazione

```
/*   * Definizione del metodo init()
      * viene effettuata la connessione al database DB2
      * tramite driver JDBC   */
  public void init()throws ServletException
   {
     // preparazione alla connessione al DB
     try {
           // definisco il driver JDBC di comunicazione con DB2
           Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
           // creazione connessione al DB
           String url = "jdbc:db2:H_Temp";
           String username = "admin";
           String passname = ”password";
           con = DriverManager.getConnection(url, username,
                     passname);
             }
```

52

Esempio inizializzazione

```
catch (SQLException ex) {
    System.out.println ("\n *** SQLException caught ***\n");
    while (ex != null) {
      System.out.println ("SQLState: " +  ex.getSQLState());
      System.out.println ("Message: " + ex.getMessage() );
      System.out.println ("Vendor: " + ex.getErrorCode() );
      ex = ex.getNextException ();
      System.out.println("");
                          }
       }
       catch (Exception e) {
              e.printStackTrace();
            }
  } // end init()
```


Distruzione servlet

- Le servlet sono attive finché il server non le rimuove tramite il metodo `destroy()`
- Ogni servlet deve ridefinire tale metodo, contenuto nella classe `HttpServlet`
- Alcuni server eseguono il metodo dopo un certo numero di secondi (server dependent)
- In caso di interazione con il DBMS, il metodo `destroy` può essere utilizzato per chiudere la connessione con il DBMS

Esempio distruzione

```
/** * Definizione del metodo destroy()
    * Effettua la chiusura della connessione al DB al termine
    * delle operazioni quando il server rimuove la Servlet */
public void destroy()
 {
  try {  con.close();     }
   catch (SQLException ex) {
     System.out.println ("\n *** SQLException caught ***\n");
     while (ex != null) {
       System.out.println ("SQLState: " + ex.getSQLState() );
       System.out.println ("Message: " + ex.getMessage() );
       System.out.println ("Vendor: " + ex.getErrorCode() );
       ex = ex.getNextException ();
       System.out.println("");
            }}
   } // end destroy()
```

55

Esecuzione richieste

- Nel periodo che intercorre tra la fase di inizializzazione di rimozione, la servlet esegue dei servizi di interazione con il client
- questi servizi sono implementati ridefinendo i metodi che gestiscono le interazioni attraverso il protocollo HTTP
- I metodi più comuni sono:
  - `doGet()` per gestire le richieste di tipo GET da pagine HTML
  - `doPost()` per gestire le richieste di tipo POST da pagine HTML

56

Esecuzione richieste

- I metodi `doGet()` e `doPost()` prevedono due argomenti di tipo:
  - `HttpServletRequest req`
  - `HttpServletResponse res`
- Tipico flusso di esecuzione in presenza di connessione a DB:
  - si imposta il file di risposta, specificandone il tipo
    `res.setContentType("text/html");` file html
  - si specifica e si inizializza la risposta da inviare al client
    `PrintWriter toClient = res.getWriter();`
  - si recuperano i parametri inviati tramite la form
    `String user = req.getParameter("user");`

57

Esecuzione richieste

- Si inviano le richieste al DBMS tramite JDBC
- si crea la pagina HTML di risposta (supponiamola contenuta nella stringa `page`)
- si invia la pagina al client
  `toClient.println(page);`
- si chiude Writer
  `toClient.close();`

58

Esempio

- Si supponga che la servlet validate venga utilizzata per verificare l'esistenza di un utente nella base di dati

59

Esempio: Form

```
...
<form action= http://concerto.disi.unige.it/servlet/validate
                                          method="POST">
 <table border="0">
   <tr>
      <td>Nome Utente</td>
        <td><input type="text" size="25" name="user"></td>
  </tr>
   <tr>
      <td>Codice Utente</td>
        <td><input type="text" size="25" name="pass"></td>
   </tr>
    <tr>
      <td><input type="submit" value="OK" border="0"></td>
   </tr>
    </table>
</form>
...
```

60

Esempio: metodo doPost()

```
Public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
                    throws ServletException, IOException {
  // imposta il contenuto del file di risposta
     res.setContentType("text/html");
  //si istanzia un oggetto di tipo Writer per mandare
  testo al client
    PrintWriter toClient = res.getWriter();
    try {
    // valori dei parametri della pagina HTML
    String user = req.getParameter("user");
    String pass = req.getParameter("pass");
```

61

Esempio: metodo DoPost()

```
// creazione statement tramite JDBC
Statement st = con.createStatement();
// genero l'istruzione SQL di ricerca sul DB
// dell'esistenza dell'utente
String sql = "Select UID from Utente Where UID='"+user+"'";
sql +=  " and PASS='"+pass+'";
// eseguo istruzione SQL
ResultSet result = st.executeQuery(sql);
String name = result.getString(1);
// scrittura del risultato su pagina WEB
String page = "<html>\n" +
          "<title>Validate Page</title>\n" +
           "Utente registrato\n" +
          "</html> \n";
```

62

Esempio: metodo doPost()

```
// Invio al Client una pagina Web con il risultato
// dell'operazione
toClient.println(page);
} catch(Exception e) {
  e.printStackTrace();
  toClient.println("A problem occured while recording" +
  "your answers.\n" + "Please try again."); }
  // Close the writer; the response is done.
toClient.close();
}// fine metodo doPost()
```


Vantaggi Servlet rispetto a CGI

- Prestazioni:
  - con le CGI, viene attivato un nuovo processo per ogni richiesta HTTP
  - con le servlet, viene attivato un solo processo
- Convenienza:
  - Java API per interagire convenientemente con dati HTML
- Potenza:
  - possibilità di condividere dati tra servlets
  - non possibile con CGI
- Portabilità:
  - dipende dalla scelta di Java
- Mantenibilità
  - stesso problema CGI

Servlet e bean

- Spesso, per interagire con la base di dati da una servlet, si utilizzano i **bean**
- un bean è una classe Java che segue alcune regole:
  - deve implementare un costruttore senza argomenti
    - StudenteBean()
  - deve avere metodi pubblici di accesso alle proprietà
    - getMatricola()
- per convenzione, si usa nominare la classe con il suffisso **Bean** (StudenteBean)
- i bean rappresentano quindi componenti riusabili
- non esiste una classe Bean, da estendere, si devono solo seguire le convenzioni

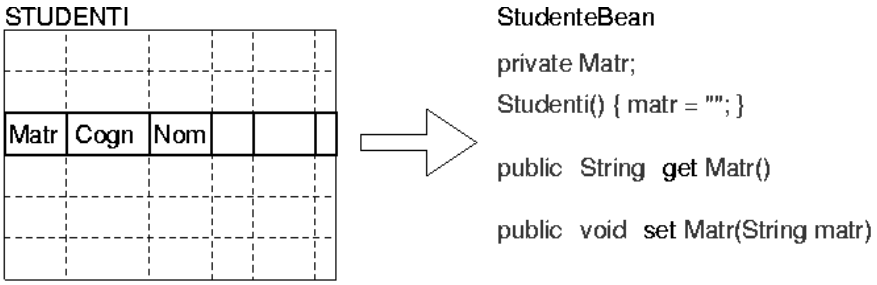
65

Esempio

```
public class StudenteBean {
    private String matricola;
    private String cognome;
    private String nome;
    /* Costruttori */                          /*  StudenteBean  */
    public StudenteBean() {
        this.matricola = "";
        this.cognome = "";
        this.nome = "";}
    /* Metodi SET */
    public void setMatricola(String matricola){this.matricola=matricola;}
    public void setCognome(String cognome) { this.cognome=cognome;}
    public void setNome(String nome) { this.nome=nome;    }
    /* Metodi GET */
    public String getMatricola() {return this.matricola;   }
    public String getCognome() { return this.cognome;    }
    public String getNome() { return this.nome;
}}
```

66

Servlet e bean



- Idea: utilizzare un bean per mappare una tupla in un oggetto Java:
  - tante variabili private quanti sono gli attributi
  - il costruttore definisce i valori di default
  - metodi pubblici getXXX e setXXX per gli attributi che si vogliono esportare

67

Esempio

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class QueryBean extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException {

        PrintWriter out = response.getWriter();
         String sql;
         Connection con = null;
         PreparedStatement pstmt;
         ResultSet rs;
```

68

Esempio

```
// parametri di connessione
String url = ” jdbc:db2:H_Temp ";
String user = ” admin ";
String passwd = ”passwd";
try {
     Class.forName("postgresql.Driver");
} catch (ClassNotFoundException cnfe) {
  System.err.println("Driver jdbc non trovato:"+cnfe.getMessage());
}
//inizializzo  HTML
     out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01
                   Transitional//EN\"");
     out.println(" "http://www.w3.org/TR/REChtml40/loose.dtd\">");
      out.println("<html>");
      out.println("<head>");
     out.println("<title>Studenti</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Studenti</h1>");
```


Esempio

```
  try {
        //Connessione, preparazione ed esecuzione statement
         con = DriverManager.getConnection(url,user,passwd);
         sql =  " SELECT *  FROM studenti WHERE matricola = ? ";
         pstmt = con.prepareStatement(sql);
         pstmt.setString(1, "IN000001");
         rs=pstmt.executeQuery();
         //istanzio Bean
         StudenteBean studente = new StudenteBean();
```

Esempio

```
//elaboro risultati query
     while (rs.next()) {
        studente.setMatr( ”IN000001");
        studente.setCognome(rs.getString("cognome"));
        studente.setNome(rs.getString("nome"));
        out.println("<p>");
        out.println("<strong>Cognome:</strong>"+studente.getCognome());
        out.println("<br>");
        out.println("<strong>Nome:</strong> "+studente.getNome());
        out.println("<br>");
        out.println("</p>");
        out.println("<hr>");
     }
     con.close();
} catch (SQLException sqle) {
     System.err.println("drivermanager non trovato: "+sqle.getMessage());
}
```

71

Esempio

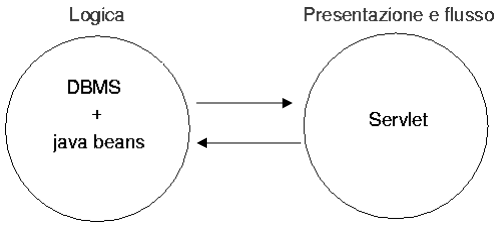
```
// termino HTML
     out.println("</body>");
     out.println("</html>");

     }
}
```

72

Servlet e bean

- I bean possono essere utilizzati per separare la logica dell'applicazione dalla parte di presentazione delle informazioni e controllo del flusso



- Evitano l'uso di una variabile per ogni attributo
  - approccio più modulare

Servlet e bean

- Per rendere ancora più indipendente la logica dal flusso, è possibile definire una classe che gestisce l'interfaccia con il DB
  - `private String studenteSql = "...";`
    il comando SQL per effettuare l'interrogazione
  - `private StudenteBean makeStudenteBean(ResultSet);`
    metodo utilizzato per popolare un bean
  - `public StudenteBean extractStudente(criterio);`
    esecuzione del comando SQL impostando il `criterio` in ingresso (es.: sostituzione chiave);
- per esercizio: provare a definire la classe (soluzione su Web)

Servlet e bean: osservazione

- Non sempre il mapping uno a uno degli attributi di una tabella in un bean risulta essere il migliore approccio
- nel caso di join tra più tabelle può essere conveniente includere nel bean anche il valore (o i valori) provenienti da entrambe le tabelle
- Tutto come prima, cambia solo la definizione del bean e la query da eseguire

75

I linguaggi di scripting server-side

- CGI, Java Servlet sono architetture general-purpose per eseguire applicazioni via HTTP e sono complesse da programmare:
- la presentazione è prodotta dal codice stesso:
  - i programmatori devono scrivere del codice per produrre HTML
  - gli esperti di HTML devono conoscere il linguaggio utilizzato (es. Java)
- Attualmente in alternativa a questi approcci sono disponibili i cosiddetti linguaggi di scripting lato server (Server-Side) che permettono di generare pagine dinamiche
- Una pagina dinamica è costituita da HTML e da codice compreso tra tag speciali (<% e %>)
  - chiara separazione tra contenuto e presentazione
- Esistono molti linguaggi di scripting come : ASP, PHP, Active Perl, ecc.
- Il Web Server deve essere esteso in modo da riconoscere le pagine dinamiche e possa inviarle al motore di scripting (Script Engine) che le interpreta e le esegue restituendo il risultato dell'elaborazione

76

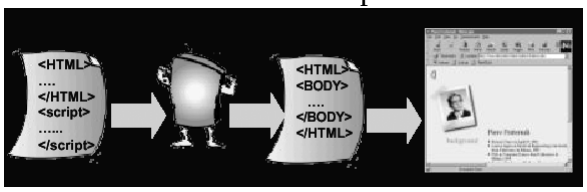
Differenze tra scripting client/server side

Il codice script è interpretato dal browser

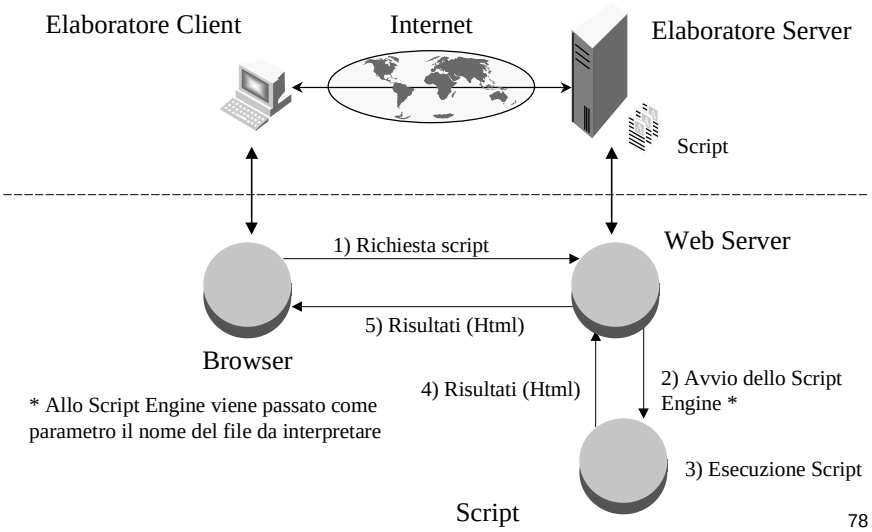
CLIENT

Il codice script è interpretato dal server
il browser riceve HTML puro

SERVER



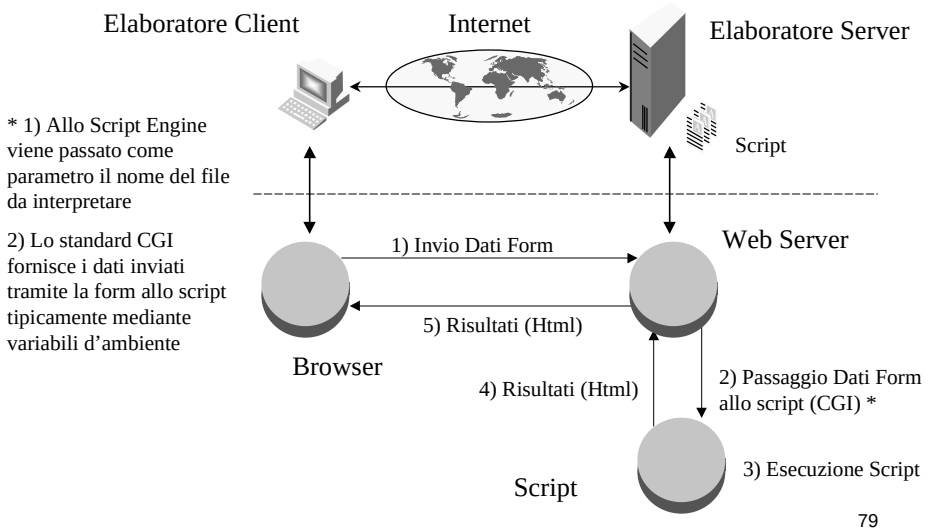
Il funzionamento di base



* Allo Script Engine viene passato come parametro il nome del file da interpretare

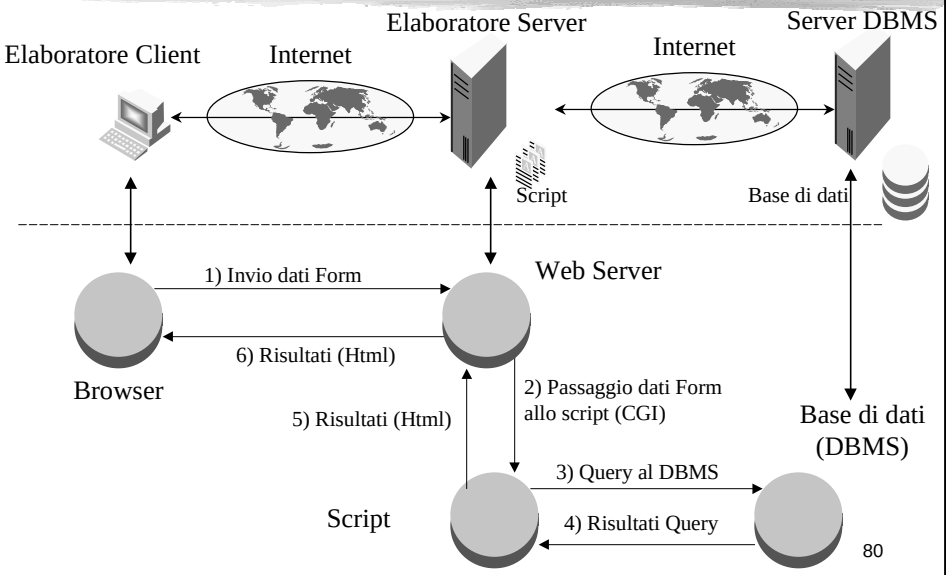
78

Il funzionamento con le form



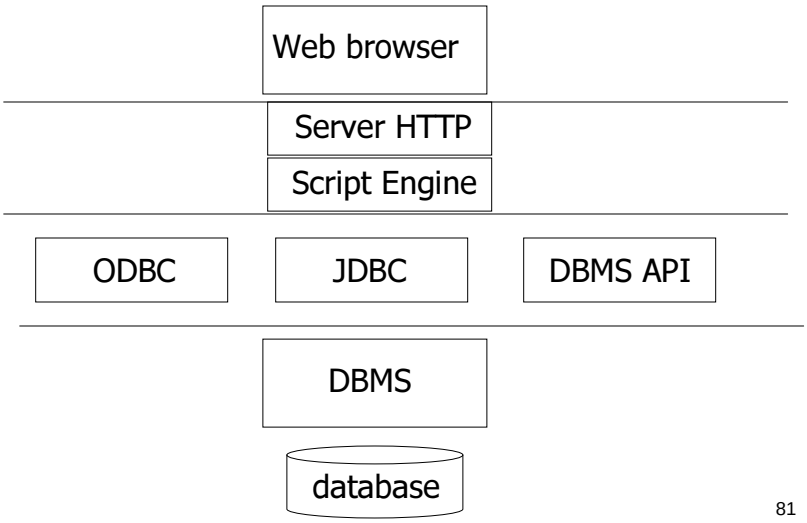
79

Interazione con il DBMS



80

Livelli



81

PHP

- PHP (Personal Home Page) è un linguaggio di scripting Server-Side
- Lo scopo di questo linguaggio è quello di facilitare lo sviluppo di pagine Web Dinamiche (on the fly)
- La sua sintassi deriva dal C, Java e Perl
- Sito ufficiale: WWW.PHP.NET
- Caratteristiche:
  - portabile (indipendente dalla piattaforma utilizzata)
  - sintassi semplice ed espressiva
  - supporto per i database più noti
  - interpretato

82

Le pagine in PHP

- Uno script o semplicemente una pagina in PHP è un file con estensione .php composta da:
  - Tag html
  - codice di programmazione PHP delimitato entro i Tag <?php e ?>

```
<HTML>
<BODY>
<?php  echo “ciao Mondo !!!”;  ?>
</HTML>
</BODY>
```

83

ASP

- ASP (Active Server Pages) è un linguaggio di scripting Server-Side
- Soluzione di Microsoft per creare pagine Web dinamiche
- Sito ufficiale: WWW.MICROSOFT.COM
- Caratteristiche:
  - non è portabile
  - indipendenza dal tipo di browser utilizzato
    - il browser vede solo pagine HTML
    - l’utente non ha bisogno di programmi proprietari o estensioni del browser
  - sintassi semplice (insieme limitato di statement Visual Basic)
  - nasconde la presenza di script agli utenti
  - supporto ODBC per i maggiori DBMS
  - interpretato

84

Le pagine in ASP

- Le Active Server Pages (ASP) sono inserite nei file con estensione .asp.
- Un file .asp è un file testuale che contiene i seguenti elementi:
  - Tag HTML
  - Codice VBScript racchiuso tra Tag speciali
    - <% ... %> comandi server-side scripting
    - <% =... %> espressioni da valutare
- Esempio 1:

```
<%  If Time >=#12:00:00 AM# And Time < #12:00:00
                PM# Then
                cSaluto = "Buon Giorno!"
   Else
     cSaluto = "Buona Sera!"
   End If
   %>
```

- Esempio 2: <%= cSaluto %>

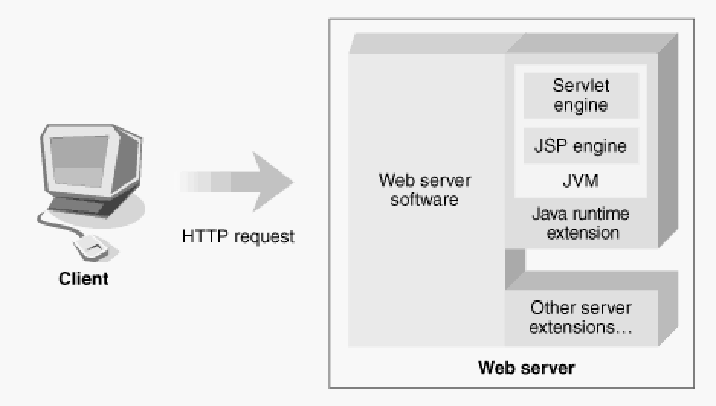
85

JSP

- JSP è una architettura per server-side scripting proposta da Sun come alternativa a ASP
- Si fonda su tecnologia Java: linguaggio Java, Java Servlet, Java Beans
- Tecnologia meno matura di ASP
- Caratteristiche:
  - portabile (indipendente dalla piattaforma utilizzata)
  - sintassi HTML + Java
  - supporto JDBC
  - compilato

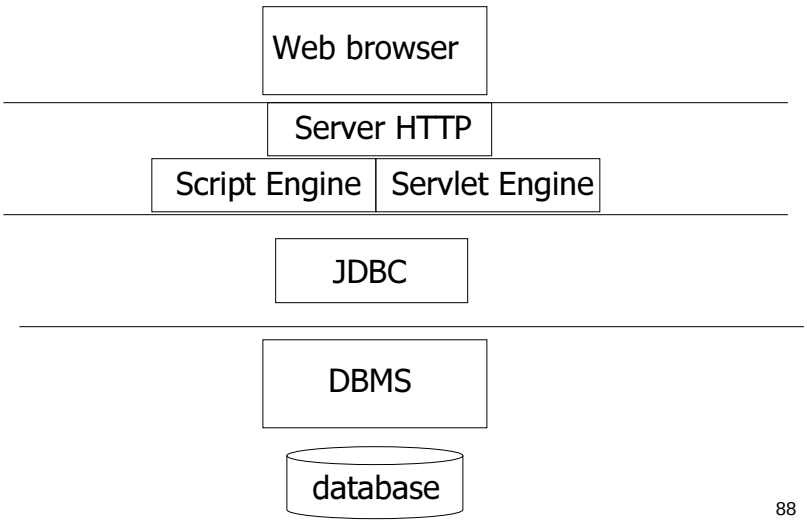
86

Architettura



87

Livelli



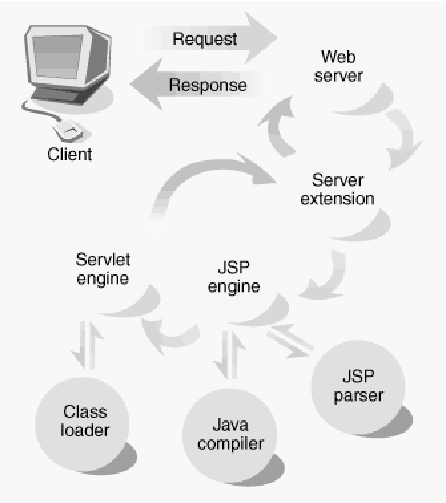
88

Architettura

- La tecnologia JSP rappresenta un livello costruito sopra alla tecnologia servlet
- una pagina JSP contiene:
  - codice HTML
  - codice Java incluso in tag specifici
- Esecuzione in quattro passi:
  1. il JSP engine parserizza la pagina e crea un file sorgente Java
  2. il file viene compilato in un class file Java, che in realtà è una servlet
  3. il servlet engine carica la servlet per l'esecuzione
  4. la servlet viene eseguita e restituisce i risultati

89

Architettura



90

Architettura

- I passi 1 e 2 vengono eseguiti al primo utilizzo della pagina e ad ogni aggiornamento
- il passo 3 viene eseguito solo in relazione alla prima richiesta della servlet
- il passo 4 può essere eseguito più volte, in relazione a quanto la pagina è dinamica

91

Sintassi (sottoinsieme)

- Dichiarazione
  - `<%! dichiarazione; [ dichiarazione; ]+ ... %>`
  - Si inseriscono dichiarazioni di **variabili di classe** (comuni a più istanze) o **metodi statici** (possono essere chiamati senza richiedere l'accesso ad una istanza, ad esempio: `nomeClasse.nomeMetodo());`
  - Es.: `<%! private int x = 4; %>`
- Espressione
  - permette di inserire valori Java nell'output
  - `<%= espressione %>`
  - Viene valutata l'espressione e stampato il risultato nella pagina html generata;
  - Es.: `<%= StudenteBean.getMatr() %>`

92

Sintassi

- Scriptlet
  - Permette di inserire codice java arbitrario nel metodo della servlet che verrà utilizzato per generare la pagina HTML finale
  - `<% codice su una o più linee %>`
  - Frammento di codice Java che può modificare anche il flusso del codice html generato.
  - Solitamente gli **operatori condizionali** (if, ?, ..) ed i **cicli** (for, while, ..) possono essere utilizzati per produrre dinamicamente porzioni diverse di codice html in funzione dell'input (analogamente alle servlet)

93

Esempio

- Servlet JSP

```
for (int i=0; i<10; i++) {
  out.println(i+": ");
  if (i%2==0) {
        out.println("<strong>pari</strong>");
  } else {
        out.println("<strong>dispari</strong>");
  }
  out.println("<br>");
}
```

- JSP

```
<% for (int i=0; i<10; i++) { %>
  <%= i %> :
  <% if (i%2==0) { %>
        <strong>pari</strong>
  <% } else { %>
        <strong>dispari</strong>
  <% } %>
  <br>
<% } %>
```

94

Sintassi

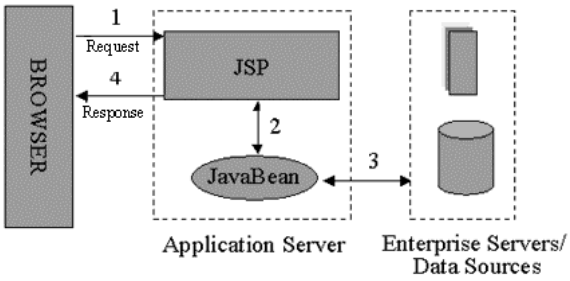
- Direttive
  - permettono di influenzare il flusso della servlet creata a partire dalla pagina JSP
  - `<%@ directive attribute1="value1" ... attribute1="valuen" %>`
- Direttiva `page`:
  - attributi:
    - `import = "package class"`
      `import ="java.util.*"`
    - `errorpage = "file"`
      `file da restituire al client se si verifica un errore`
- Direttiva `include`: permette di includere dei file nel momento in cui la pagina JSP è tradotta in una servlet
  - Attributo: `file = "url"`

Sintassi

- Esistono tag speciali per interagire con i bean
  - non li vediamo per mancanza di tempo
- In ogni caso, è possibile utilizzare i bean direttamente con codice Java

Modello di sviluppo: JSP/1

- Il client richiede via HTTP un file .JSP
- Il file .JSP viene interpretato e accede a componenti lato-server (tipicamente Java Beans, Servlet) che generano contenuti dinamici
- il risultato viene spedito al client sotto forma di pagine HTML



97

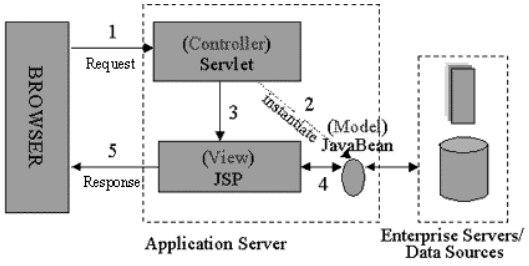
Modello di sviluppo: JSP/1

- Vantaggi:
  - se si usano i beans, chiara separazione tra presentazione e logica
  - ottimo per applicazioni di piccole dimensioni
- Svantaggi:
  - molto codice Java dentro pagine JSP
  - problemi sviluppo e mantenibilità

98

Modello di sviluppo: JSP/2

- Uso combinato di servlet e JSP
  - JSP per livello di presentazione
  - servlet per accesso DB tramite beans
- La pagina JSP è incaricata di recuperare i bean creati dalla servlet, estrarne il contenuto per inserirlo nelle pagine statiche



99

Modello di sviluppo JSP/2

- Flusso tipico:
  - Servlet:
    - riceve richiesta dal browser
    - determina operazione da eseguire
    - interagisce con il DBMS e crea eventuali bean
    - ➨ rende disponibili i bean creati alla pagina JSP per la visualizzazione
    - ➨ attiva la pagina JSP per la visualizzazione
  - pagina JSP:
    - ➨ accede ai bean creati dalla servlet
    - ➨ utilizza le informazioni contenute nei bean per creare la pagina da visualizzare (quindi da restituire al client)

100

Servlet: accesso ad altre risorse

- Le servlet possono accedere ad altre risorse (es. pagine JSP) in due modi:
  - tramite protocollo HTTP (problematica programmazione Java)
  - tramite una particolare richiesta, se la risorsa risiede sul server in cui la servlet viene eseguita
- Tutte le servlet in esecuzione sullo stesso server formano un *contesto*
  - L'oggetto contesto è recuperabile con `getServletContext()`
- Due problemi:
  - come attivare una comunicazione con un'altra risorsa, a partire da un certo contesto
  - come forwardare la richiesta del client alla nuova risorsa (è anche possibile non forwardarla ma permettere una risposta parziale, non lo vediamo)

Servlet: accesso ad altre risorse

- **Attivazione comunicazione**:
  - si crea un oggetto `RequestDispatcher` a partire dal contesto

```
RequestDispatcher dispatcher
getServletContext().getRequestDispatcher(nome risorsa);
```

  - `nome risorsa` identifica il nome del file corrispondente, a partire da una directory che dipende dal contesto (cioè dal Web server)
  - Esempio:

```
RequestDispatcher dispatcher
getServletContext().getRequestDispatcher
                          ("/jsp/visStud.jsp");
```

Servlet: accesso ad altre risorse

- **Forward richiesta client:**
  - dopo avere creato l'oggetto `RequestDispatcher`, si può associare alla risorsa corrispondente la responsabilità di rispondere alla richiesta del client

    ```
    dispatcher.forward(req,res)
    ```

    dove `req` e `res` sono di tipo `HttpServletRequest` e `HttpServletResponse`, rispettivamente

  - a questo punto spetta alla risorsa inviare l'output al client

103

Servlet: condivisione risorse

- Le servlet in uno stesso contesto possono condividere risorse, in forma di attributi, usando l'interfaccia associata al contesto
- associando un bean ad un attributo, rendiamo disponibile il bean a tutte le servlet del contesto

```
StudenteBean studente = new StudenteBean();
getServletContext().setAttribute("studente",studente);
getServletContext().getAttribute("studente",studente);
```

104

Servlet: condivisione risorse

- Lo stesso approccio si può utilizzare per condividere informazioni con altre risorse
- in questo caso, gli attributi verranno associati non più al contesto ma alla richiesta che dovrà essere inviata alla risorsa

```
StudenteBean studente = new StudenteBean();
req.setAttribute("studente",studente);
…
dispatcher.forward(req,res)
```

- se la risorsa a cui mandiamo la richiesta è una pagina JSP, allora la pagina potrà accedere il bean precedentemente creato dalla servlet

105

JSP: accesso ai bean

- I bean in JSP possono essere facilmente manipolati utilizzando le *azioni*
- le azioni usano una sintassi XML
- le azioni significative a livello di manipolazione bean sono tre:
  - `jsp:useBean`: per creare o rendere disponibile un bean in una pagina JSP
  - `jsp:setProperty`: per settare proprietà di un bean
  - `jsp:getProperty`: per recuperare valori associati alle proprietà di un bean

106

JSP: useBean

- `<jsp:`**`useBean`** `id="name" class= "package.class" scope= "scope"/>`
- permette di creare un'istanza della classe `package.class`, chiamata `name`
- l'istanza viene resa disponibile in un certo contesto, specificato da `scope:`
  - `page:` all'interno della pagina JSP
  - `request:` all'interno della richiesta client corrente (la richiesta viene identificata dalla variabile di sistema `request`)
  - …
- se nel contesto esiste già un bean con il nome prescelto, allora il bean non viene creato ma reso disponibile alla pagina
- Esempio:

```
<jsp:useBean id="studente" class= "StudenteBean"
             scope= "request"/>
```

107

JSP: setProperty, getProperty

- `<jsp:`**`setProperty`** `name="name" property= "attributeName" value= "valore"/>`
  - Esempio:

```
<jsp:setProperty name="studente" property= "cognome"
                 value= "Rossi"/>
```

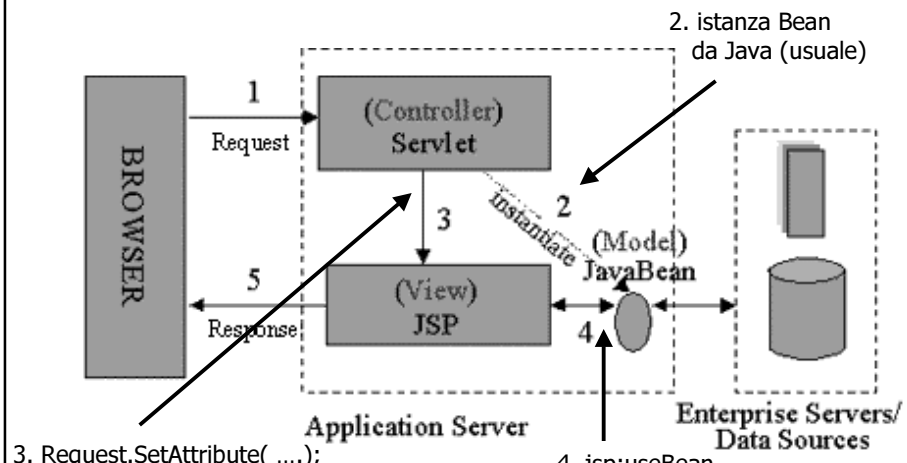
- `<jsp:`**`getProperty`** `name="name" property= "attributeName" />`
  - Esempio:

```
<jsp:getProperty name="studente"
                 property= "cognome" />
```

  - il tag restituisce un'espressione, rappresentata dal valore del bean associato all'attributo specificato

108

Modello di sviluppo: JSP/2



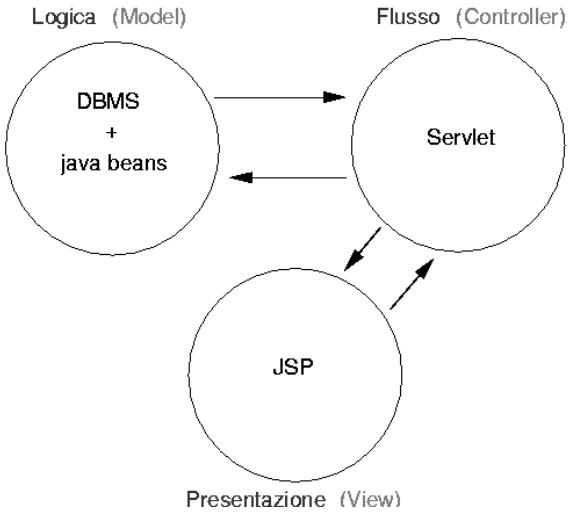
3. Request.SetAttribute(….);

RequestDispatcher dispatcher = …
dispatcher.forward(Request,Response)

4. jsp:useBean ...
jsp:setProperty ...
jsp:getProperty ...

109

Modello di sviluppo: JSP/2



110

Modello di sviluppo: JSP/2

- Vantaggi:
  - netta separazione tra logica, flusso e presentazione
  - ottimo per applicazioni di medie/grandi dimensioni
- Svantaggi:
  - sistema più complesso da progettare
- Per chi è interessato:
  - su web applicazione completa

111

Vantaggi JSP

- Rispetto ad ASP:
  - portabilità (grazie a Java)
  - indipendenza dall'ambiente
- rispetto alle servlet:
  - niente di più in termini di potere espressivo
  - sviluppo e mantenibilità più semplice

112

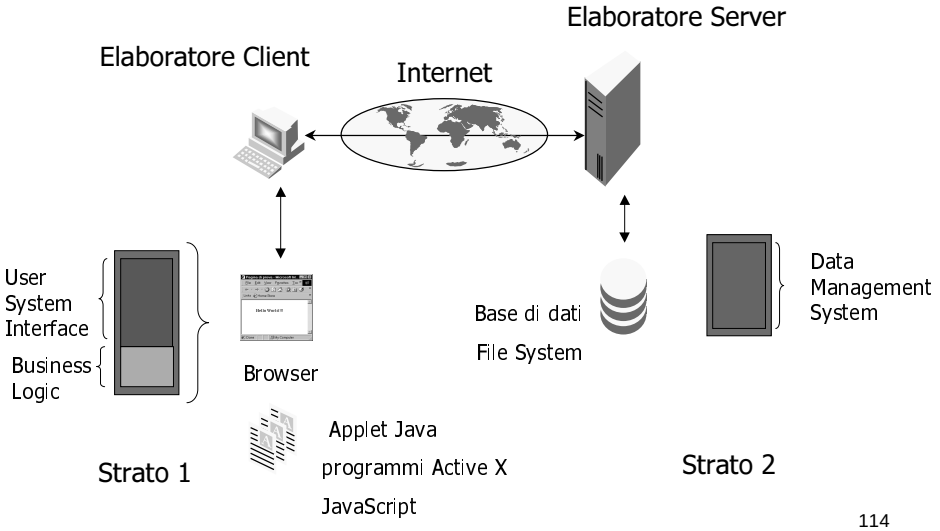
Soluzioni lato client

Vedremo:

- **Applet**

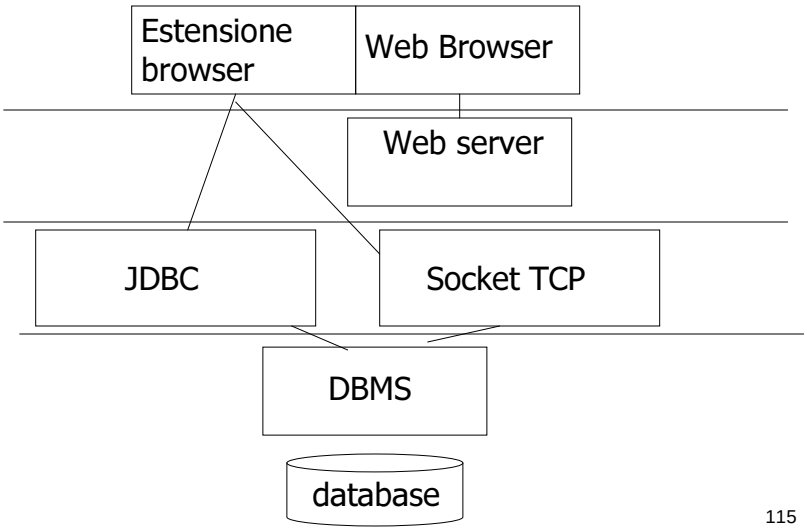
113

Sistema classico a 2-strati (lato client)



114

Livelli



115

Estensione al browser

- Le tecnologie lato client si possono tutte vedere come approcci che estendono le funzionalità del browser:
  - tramite applicazioni Java
    - Applet
  - tramite script
    - JavaScript
    - VBScript
- In entrambi i casi, è possibile comunicare con il DBMS o tramite protocolli di comunicazione (JDBC da Applet) o direttamente tramite sockets

116

Vantaggi e svantaggi

- Vantaggi
  - permette in modo semplice di manipolare i dati inseriti dall'utente
  - facile da usare
  - flessibile
- Svantaggi
  - carica di lavoro il client
  - problematiche di sicurezza
  - tipicamente usata per gestire l'interfaccia e non per gestire la logica applicativa
  - meno usate e flessibili rispetto alla tecnologia lato server

117

Java Applet

- Un applet è un programma Java che viene scaricato dalla rete e viene eseguito dal browser
- L'applet è collegata ad una pagina Web tramite un link al programma Java corrispondente
- Quando si carica la pagina si carica il codice eseguibile (bytecode) del programma
- Dopo essere stato caricato l'applet viene mandato in esecuzione sul client dentro la pagina HTML
- Risultato
  - Si estende senza limiti la funzionalità del Web!

118

Problemi

- Problema:
  - esecuzione sulla propria macchina di un codice proveniente da una fonte ignota (non affidabile in linea di principio)
- Soluzione:
  - linguaggio fortemente tipato, senza puntatori
  - un Applet non può accedere al file system locale
  - non può aprire connessioni di rete con host diversi da quello di provenienza
  - non può lanciare altri programmi su host diversi da quello di provenienza

Ciclo di vita

- Un applet si definisce creando una sottoclasse Java della classe predefinita `Applet`
- In genere un applet ridefinisce i seguenti metodi:
  - `init()`: inizializza l'applet ogni volta che viene caricata
  - `start()`: codice eseguito al momento del caricamento dell'applet e ogni volta che si visita la pagina ad essa collegata
  - `stop()`: termina l'esecuzione dell'applet, quando si lascia la pagina collegata all'applet
  - `destroy()`: pulizia prima di scaricare l'applet

Applet e HTML

- In una pagina HTML si fa riferimento ad un applet con uno specico tag:
  - `<APPLET CODE="es.class" WITDH=50 HEIGHT=50></APPLET>`
  - Il client riserva uno spazio di 50x50 pixel nella pagina per l'esecuzione dell'Applet (output, eventi, ecc.).
- L'Applet viene fisicamente collocata:
  - Nella stessa directory del file html che la carica
    - `<APPLET CODE="es.class" WIDTH=...`
  - In una sottodirectory
    - `<APPLET CODE="es.class" CODEBASE="examples/" ...`
  - In un altro sito
    - `<APPLET CODE="es.class"`
      `CODEBASE="http://someserver/somedir" ...`

121

Applet e DBMS

- In quanto applicazione Java, l'applet può comunicare con un DBMS tramite JDBC
  - Unico problema: il DBMS deve risiedere sull'host dal quale proviene l'applet
- L'applet può anche invocare servlet o programmi CGI, che risiedono sull'host dal quale proviene l'applet, tramite socket
  - questi programmi possono a loro volta connettersi ad un DBMS remoto
  - applet: gestione interfaccia
    servlet: gestione logica applicativa
  - soluzione che permette di distribuire il carico di lavoro tra client e server

122

Riassumendo ...

123

Architettura

- L'interazione basi di dati e Web richiede:
  - A livello fisico:
    - l'introduzione di un ulteriore livello nella tipica architettura client/server del Web
    - necessità di gateway per comunicare con il DBMS
      - JDBC, ODBC, DBMS API
  - A livello logico:
    - chiara distinzione tra flusso, logica e presentazione
    - JavaBeans
    - componenti ActiveX nel contesto Microsoft

124

Aspetti

- Per interagire con una base via Web è necessario tenere presente i seguenti aspetti:
  - tecnologia client side rispetto a tecnologia server side
    - è consigliata la tecnologia server side
    - tecnologia client side solo per gestire l'interfaccia
  - programmi compilati o script
    - la soluzione migliore combina entrambe le tecnologie
      - script per presentazione
      - programmi per logica applicativa

125