



Evoluzione del modello: Web dinamico

Su Virtuale:

Versione 1 pagina per foglio = 2.01.WebDinamico.pdf

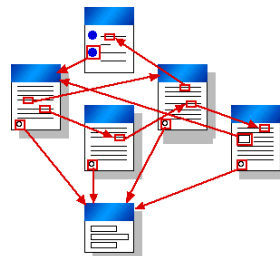
Versione 2 pagine per foglio = 2.01.WebDinamico-2p.pdf

1

1

Modello Web statico

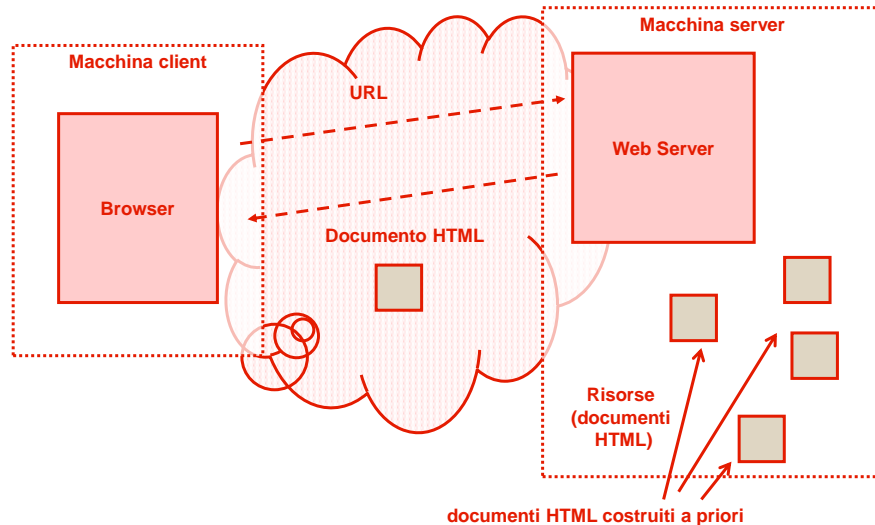
- Il modello che abbiamo analizzato finora, basato sul concetto di ipertesto distribuito, ha una **natura essenzialmente statica**
- Anche se l'utente può percorrere dinamicamente l'ipertesto in modi molto diversi, **l'insieme dei contenuti è prefissato staticamente**:
 - **Pagine** vengono **preparate staticamente** a priori
 - Non esistono contenuti composti dinamic. in base all'interazione con l'utente
- È un modello semplice, potente, di facile implementazione efficiente, ma presenta evidenti limiti



2

2

Modello Web Statico

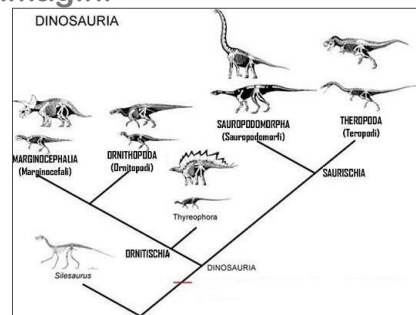


3

3

Limiti del modello statico

- Per capire quali sono i limiti del modello statico e come possono essere superati proviamo a ragionare su un semplice esempio
- Vogliamo costruire un'enciclopedia dei Dinosauri consultabile via Web: <http://www.dino.it>
 - Possiamo creare *una pagina HTML per ogni specie* di dinosauro con testi e immagini
 - Possiamo poi creare una *pagina iniziale* che fa da *indice* basandoci sulla classificazione scientifica
 - *Ogni voce è un link* alla pagina che descrive un dinosauro



4

4

Limiti del modello statico

- Se vogliamo rendere più agevole l'accesso alle schede possiamo anche predisporre un'altra pagina con un *indice analitico* che riporta le *specie di dinosauri in ordine alfabetico*
- Tutto questo può essere realizzato facilmente con gli strumenti messi a disposizione dal Web statico
- Il modello va però *in crisi* se proviamo ad *aggiungere una funzionalità molto semplice: ricerca per nome*
- Quello che ci serve è una *pagina con un form*, costituito da un semplice campo di input e da un bottone, che ci consenta di inserire il nome di un dinosauro e di accedere direttamente alla pagina che lo descrive

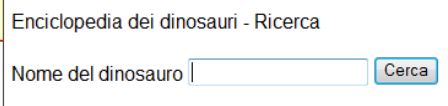
5

5

Ricerca

Vediamo il codice della pagina HTML: abbiamo usato il metodo GET per semplicità

```
<html>
<head>
  <title> Ricerca dinosauri </title>
</head>
<body>
  <p>Enciclopedia dei dinosauri - Ricerca</p>
  <form method="GET" action="http://www.dino.it/cerca">
    <p>Nome del dinosauro
    <input type="text" name="nomeTxt" size="20">
    <input type="submit" value="Cerca" name="cercaBtn">
    </p>
  </form>
</body>
</html>
```

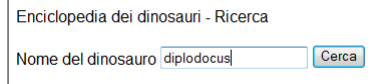


6

6

Eseguire la ricerca

- Se scriviamo il nome di un dinosauro e premiamo il bottone cerca, si attiva una invocazione HTTP di tipo GET con un URL di questo tipo:



Enciclopedia dei dinosauri - Ricerca

Nome del dinosauro

`www.dino.it/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`

- Web server non è in grado di interpretare immediatamente questa chiamata (URL con query) *perché richiede l'esecuzione dinamica di un'applicazione* legata al particolare contesto
- *È quindi necessaria un'estensione specifica:* un programma scritto appositamente per l'enciclopedia che
 - Interpreti i parametri passati nel GET
 - Cerchi nel file system la pagina `diplodocus.html`
 - La restituisca al Web server per l'invio al client

7

7

CGI

- La prima soluzione proposta per risolvere questo problema prende il nome di **Common Gateway Interface (CGI)**, già presente fino da HTTPv1.0
- *CGI è uno standard per interfacciare applicazioni esterne con Web server*
- Le applicazioni che usano questo standard prendono il nome di **programmi CGI**
- *Un programma CGI viene eseguito dinamicamente* in risposta alla chiamata e produce output che costituisce la risposta alla richiesta http (**informazione dinamica**)
- Può essere scritto in qualunque linguaggio: ad esempio in C o in un linguaggio di script (spesso PHP o Perl) o in un qualche linguaggio ibrido (es. Python)

8

8

Modello di interazione CGI



Le operazioni si svolgono nel seguente ordine:

- Il client, tramite HTTP, invia al server la richiesta di eseguire un programma CGI con alcuni parametri e dati in ingresso
- Il server, *attraverso l'interfaccia standard CGI (accordo standardizzato)*, chiama il programma passandogli i parametri e i dati inviati dal client
- Eseguite le operazioni necessarie, il programma CGI rimanda al server i dati elaborati (pagina HTML), sempre facendo uso dell'interfaccia CGI
- Il server invia al client i dati elaborati dal programma CGI tramite protocollo HTTP

9

9

Comunicazione fra server e programma CGI

I programmi CGI e il server comunicano in quattro modi (specificati nell'interfaccia standard CGI):

- **Variabili di ambiente** del sistema operativo
- **Parametri sulla linea di comandi**: programma CGI viene lanciato in un processo pesante (si pensi a shell di sistema operativo che interpreta i parametri passati, ad esempio in metodo GET)
- **Standard Input** (usato con il metodo POST)
- **Standard Output**: per restituire al server la pagina HTML da inviare al client

10

10

Parametri: metodo GET

- *Con il metodo GET il server passa il contenuto della form al programma CGI come se fosse da linea di comando di una shell*
- Nel nostro esempio la URL era
`www.dino.it/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`
- Dove:
 - `www.dino.it` è l'indirizzo del server web
 - `cerca` è il nome del programma CGI
 - `nomeTxt=diplodocus&cercaBtn=Cerca` è la riga di comandi passata al programma cerca
- Linea di comando ha una lunghezza finita dipendente da SO (massimo 256 caratteri su SO UNIX)
- Quindi, come già sapete, la quantità di dati che possono essere inviati con il metodo GET è molto limitata

11

11

Parametri: metodo POST

- Come abbiamo già visto, **usando POST non viene aggiunto nulla alla URL specificata da ACTION**
- Quindi la linea comando nella shell contiene solo il nome del programma CGI
- Nel nostro esempio la URL sarà semplicemente `www.dino.it/cerca` a cui corrisponde un comando di linea `cerca` senza alcun parametro
- I dati del form, contenuti nell'header HTTP, vengono inviati al programma CGI tramite ***standard input***
- In questo modo ***si possono inviare dati lunghi a piacimento***, senza i limiti di GET
- In C per accedere ai dati si apre un file su `stdin` e si leggono i campi del post con `fgetc()`:

```
nome = fgetc(stdin); // nomeTxt=diplodocus
btn = fgetc(stdin); // cercaBtn=Cerca
```

12

12

Variabili di ambiente

Prima di chiamare il programma CGI, Web server imposta *alcune variabili di sistema corrispondenti ai principali header HTTP*, ad esempio:

- **REQUEST_METHOD**: metodo usato dalla form
- **QUERY_STRING**: parte di URL che segue il "?"
- **REMOTE_HOST**: host che ha inviato la richiesta
- **CONTENT_TYPE**: tipo MIME dell'informazione contenuta nel body della richiesta (nel POST)
- **CONTENT_LENGTH**: lunghezza dei dati inviati
- **HTTP_USER_AGENT**: nome e versione del browser usato dal client

Se si implementa CGI in C, si può usare **getenv()**:

```
utente = getenv(REMOTE_HOST);
```

13

13

Output

- Il programma CGI elabora i dati in ingresso ed emette un output per il client in attesa di risposta
- *Per passare i dati al server il programma CGI usa **stdout***: ad es. in C si può usare la funzione **printf()**
- Il server preleva i dati dallo standard output e li invia al client *incapsulandoli in messaggio HTTP*, ad esempio:

```
HTTP/1.0 200 OK
Date: Wednesday, 02-Feb-94 23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Monday, 15-Nov-93 23:33:16 GMT
Content-type: text/html
...
<HTML><HEAD><TITLE>
...
```

Deve contenere almeno
header content-type; altri
opzionali...

Parte generata
da Web server

Parte generata
dal programma
CGI

14

14

Configurazione Web server

- Se arriva URL `www.dino.it/cgi-bin/cerca` server deve rendersi conto che `cerca` non è un documento HTML ma un *programma CGI*
- Perché ciò accada è necessario che:
 - I programmi CGI siano *tutti in un'apposita directory*
 - Nella configurazione del server sia specificato il *path ove trovare i programmi CGI* e l'identificatore che indica che è richiesta l'esecuzione di una applicazione
- Di solito si sceglie come identificatore `/cgi-bin/`
- Tutto ciò che segue l'identificatore viene interpretato dal server come nome di programma da eseguire
- Il server cerca il programma specificato nel path che è stato indicato nella configurazione

15

15

CGI e dinosauri

- Un meccanismo come quello appena descritto consente di risolvere il problema della ricerca nell'esempio dell'enciclopedia
- Il browser usando il metodo GET invia il contenuto del campo di ricerca nella parte query dell'URL:
`www.dino.it/cgi-bin/cerca?nomeTxt=diplodocus&cercaBtn=Cerca`
- Web server usa l'interfaccia CGI per passare i parametri presenti nell'URL ad un programma denominato `cerca`
- `cerca` usa il valore del parametro `nomeTxt` per cercare le pagine che contengono il termine inserito nel campo di ricerca (`diplodocus`)
- Usa `stdout` per costruire una pagina con un elenco di link alle pagine che contengono il termine

16

16

Altri problemi

- La nostra enciclopedia ha anche un problema di manutenibilità
- Se vogliamo aggiungere un dinosauro dobbiamo infatti:
 - Creare una nuova pagina con una struttura molto simile alle altre
 - Aggiungere il link alla pagina nell'indice principale (quello basato sulla classificazione delle specie)
 - Aggiungere un link nell'indice alfabetico
- La pagina di ricerca invece non richiede alcuna modifica
- Se poi volessimo cambiare l'aspetto grafico della nostra enciclopedia dovremmo rifare una per una tutte le pagine (a meno dell'utilizzo di CSS e XHTML di cui siete oramai grandi esperti...)

17

17

DynamicDino.it

- Una soluzione ragionevole è quella di **separare gli aspetti di contenuto da quelli di presentazione**
- Ad esempio, utilizziamo un database relazionale per memorizzare le informazioni relative ad ogni dinosauro
- Realizziamo alcuni programmi CGI che generano dinamicamente l'enciclopedia
 - **scheda**: crea una pagina con la scheda di un determinato dinosauro
 - **indice**: crea la pagina di indice per specie (tassonomia)
 - **alfabetico**: crea l'indice alfabetico
 - **cerca**: restituisce una pagina con i link alle schede che contengono il testo inserito dall'utente
- Tutti e 4 i programmi usano il DB per ricavare le informazioni utili per la costruzione della pagina

18

18

Struttura del database

- Per semplicità possiamo utilizzare una sola tabella con la struttura sotto riportata (non è normalizzata)
- Il campo testo serve a includere il contenuto in formato HTML
- Gli altri campi permettono di costruire agevolmente l'indice basato sulla tassonomia

Specie	Ordine	Famiglia	Genere	Testo
Dicraeosaurus hansemani	Saurischia	Dicraeosauridae	Dicraeosaurus	<p>Il dicreosauro (gen. Dicraeosaurus) è un dinosauro erbivoro vissuto in Africa orientale nel Giurassico superiore (Kimmeridgiano, circa 150 milioni di anni fa).
...
...

19

19

Vantaggi della nuova soluzione

- In questo modo la gestione dell'enciclopedia è ***sicuramente più semplice***:
 - Basta ***inserire un record nel database*** per aggiungere una nuova specie
 - L'indice tassonomico e quello alfabetico si aggiornano automaticamente
- ***È anche possibile cambiare agevolmente layout di tutte le pagine***
 - Possibile utilizzare una pagina HTML di base (template) con tutte le parti fisse
 - Il programma scheda si limita a caricare il template e a inserire le parti variabili
- Per cambiare l'aspetto grafico di tutte le schede è sufficiente agire una sola volta sul template

20

20

Un nuovo modello

Il modello è cambiato in modo significativo

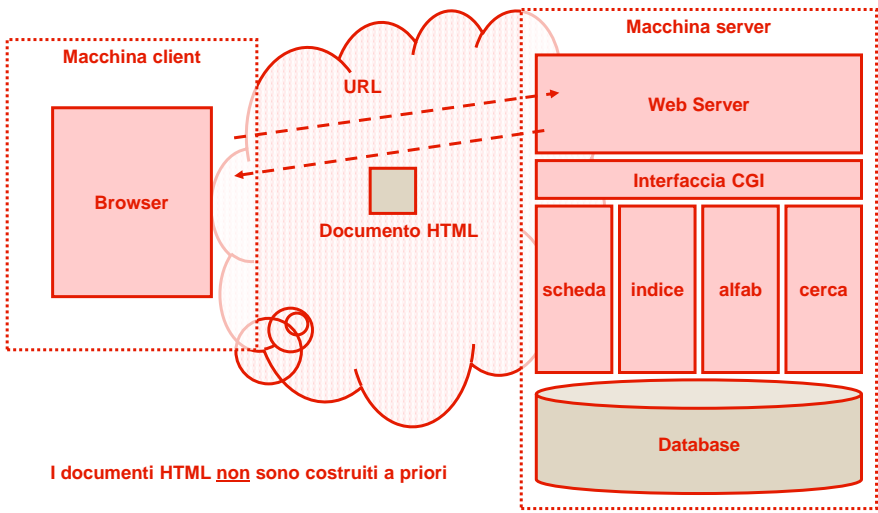
- Abbiamo aggiunto una *forma di elaborazione lato server*
- L'insieme delle **CGI** che gestiscono l'enciclopedia costituisce *un'applicazione distribuita*
- Ogni CGI può essere vista come una *procedura remota invocata tramite HTTP dal cliente*

URL (usualmente in /cgi-bin/)	Procedura
/scheda?specie=Diplodocus	scheda("Diplodocus");
/indice	indice();
/alfabetico	alfabetico();
/cerca?testo=diplod	cerca("diplod");

21

21

Modello Web dinamico



22

22

Pagine Web Statiche, Dinamiche, altro?

- **Pagine Web statiche**
 - Uso di HTML, CSS e altre risorse statiche (cacheable)
- **Pagine Web dinamiche**
 - CGI
 - Java Servlet
 - JSP, PHP, ASP, ...
- **Pagine Web attive** (ne parleremo diffusamente più avanti...)
 - Javascript lato cliente (eventualm + AJAX)

23

23

Vedi esempio ulteriore e dettagliato su Virtuale

- [studForm.html](#) come esempio di semplice form HTML
- [CGI example.c](#) come semplice esempio di programma CGI (in linguaggio C)
 - Quindi ovviamente da compilare prima del deployment su Web server

24

24

Tutto a posto? ASSOLUTAMENTE NO

L'architettura che abbiamo appena visto presenta numerosi vantaggi ma soffre anche di diversi problemi

- ***Ci sono problemi di prestazioni:*** ogni volta che viene invocata una CGI si crea un **processo** che viene **distrutto** alla fine dell'elaborazione
 - Esistono varianti di CGI che risolvono alcuni problemi di prestazioni (FastCGI). Come, secondo voi?
- Le CGI, soprattutto se scritte in C, possono essere **poco robuste** (*che cosa succede se errore bloccante?*)
- Ogni programma CGI deve **reimplementare tutta una serie di parti comuni** (*mancanza di moduli di base accessibili a tutti i programmi lato server*): accesso al DB, logica di interpretazione delle richieste HTTP e di costruzione delle risposte, gestione dello stato ecc.
- Abbiamo scarse garanzie sulla sicurezza

25

25

Una sola applicazione

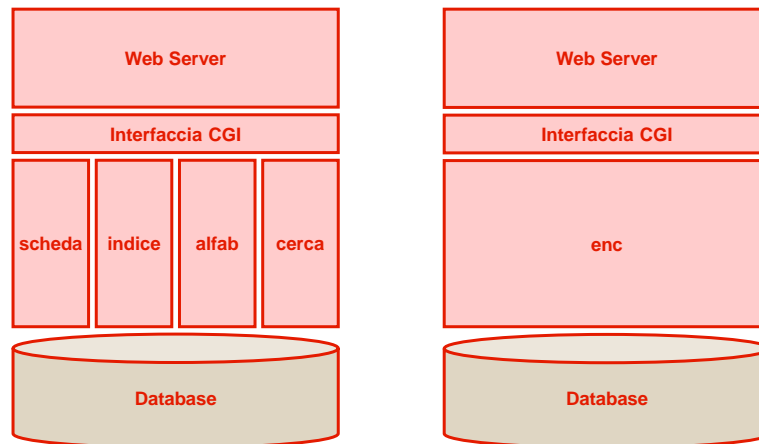
- Per ovviare al penultimo punto si potrebbe realizzare una sola CGI (**enc**) che implementa tutte e quattro le funzionalità (vedi schema sottostante)
- In questo modo però si ha **un'applicazione monolitica** e si perdono i vantaggi della modularità
- Gli altri problemi rimangono invariati

URL	Procedura
/enc?azione=scheda&specie=Diplodocus	scheda("Diplodocus");
/enc?azione=indice	indice();
/enc?azione=alfabetico	alfabetico();
/enc?azione=cerca&testo=diplod	cerca("diplod");

26

26

Le due soluzioni a confronto



27

27

Application server

- La soluzione migliore è quella di realizzare un contenitore in cui far “vivere” le funzioni server-side
- Il contenitore si preoccupa di fornire i servizi di cui le applicazioni hanno bisogno:
 - Interfacciamento con il Web Server
 - Gestione del tempo di vita (attivazione on-demand delle funzioni)
 - Interfacciamento con il database
 - Gestione della sicurezza
- Si ha così una soluzione modulare in cui le funzionalità ripetitive vengono portate a fattor comune
- Un ambiente di questo tipo prende il nome di **application server**

28

28

Parentesi su Modelli a Contenimento

Che cosa vi ricordate di modelli componenti-container?
Esempi di tecnologie a componenti?

CONTENIMENTO

Spesso molte funzionalità possono essere non controllate direttamente ma lasciate come responsabilità ad una *entità delegata supervisore* (**contenitore**) che se ne occupa

- spesso introducendo politiche di default
- evitando che si verifichino errori
- controllando eventuali eventi

I **contenitori** (entità dette anche **CONTAINER**, **ENGINE**, **MIDDLEWARE**, ...) possono occuparsi di *azioni automatiche* da cui viene sgravato l'utilizzatore che deve specificare solo la parte contenuta tipicamente

di alto livello,

non ripetitiva,

fortemente dipendente *dalla logica applicativa*

29

29

Modelli a Contenimento

CONTAINER

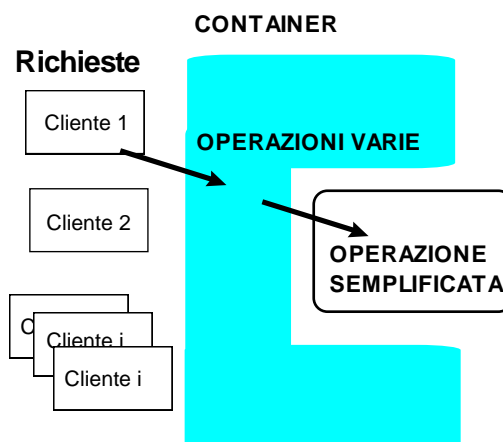
Un servizio utente potrebbe essere *integrato in un ambiente (middleware) che si occupa in modo autonomo* di molti aspetti diversi

Vedrete ad esempio

CORBA - tutti aspetti C/S
Engine per framework a GUI

Container per servlet

Container possono ospitare componenti più trasportabili, riutilizzabili e mobili



30

30

Delega al Container

Il container può fornire “*automaticamente*” molte delle funzioni per supportare il servizio applicativo verso l’utente

- **Supporto al ciclo di vita**

- Attivazione/deattivazione* del servitore

- Mantenimento dello *stato* (durata della sessione?)

- Persistenza trasparente* e recupero delle informazioni (interfaccia DB)

- **Supporto al sistema dei nomi**

- Discovery* del servitore/servizio

- Federazione* con altri container

- **Supporto alla qualità del servizio**

- Tolleranza ai *guasti*, selezione tra possibili *deployment*

- Controllo della *QoS richiesta e ottenuta*

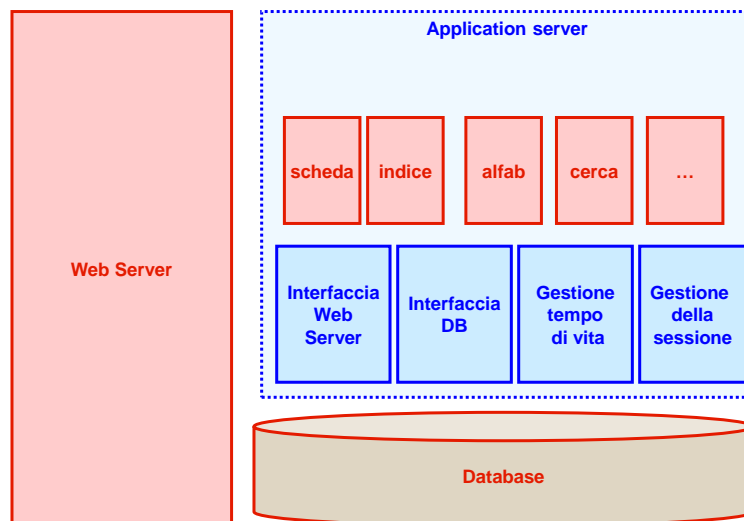
- **Sicurezza**

- ...

31

31

Architettura basata su application server



32

32

Application server e tecnologie server side

- Due tecnologie storicamente molto diffuse nell'ambito degli application server sono:
 - .NET per componenti Microsoft ed evoluzioni
 - Java J2EE per componenti EJB e Web Java-oriented
- Altre tecnologie interessanti:
 - Ad esempio quelle basate su Perl, Python, o Javascript lato server (node.js, ...)
- Altre soluzioni hanno una struttura più semplice e non sono application server a tutti gli effetti ma spesso solo interpreti (si parla di moduli di estensione del Web server – comunque interessanti per applicazioni Web a rapida prototipazione e basso costo):
 - *PHP* (molto diffuso e di semplice utilizzo) o *Ruby*
 - Le “vecchie” tecnologie *ISAPI* e *ASP* di Microsoft
 - ...

33

33

Altri aspetti: lo stato

- L'enciclopedia dei dinosauri è un'applicazione ***stateless***
- Il server e i programmi CGI non hanno necessità di tener traccia delle chiamate precedenti

L'interazione tra un Client e un Server può essere infatti di due tipi:

- **Stateful**: esiste stato dell'interazione e quindi l'n-esimo messaggio può essere messo in relazione con gli n-1 precedenti
- **Stateless**: non si tiene traccia dello stato, ogni messaggio è indipendente dagli altri

34

34

Interazione stateless

- In termini generali, un'interazione stateless è “feasible” senza generare grossi problemi solo se protocollo applicativo è progettato con **operazioni idempotenti**
- *Operazioni idempotenti producono sempre lo stesso risultato, indipendentemente dal numero di messaggi M ricevuti dal Server stesso.* Ad es. un server fornisce sempre la stessa risposta a un messaggio M
- Nel nostro caso tutte e 4 le operazioni gestite dall'enciclopedia (indice, alfabetico, scheda e cerca) sono idempotenti. In generale, molto spesso abbiamo a che fare con operazioni idempotenti nelle applicazioni Web
- Esempi di operazioni non-idempotenti?

35

35

Interazione stateful

- **Non tutte le applicazioni possono fare a meno dello stato**

Esempio banale: se è prevista un'autenticazione è molto comodo (necessario a fini di usabilità utente) tener traccia fra una chiamata e l'altra del fatto che l'utente si è autenticato e quindi nasce l'esigenza di avere uno stato

In generale, tutte le volte in cui abbiamo bisogno di *personalizzazione delle richieste Web*, possiamo beneficiare di interazione stateful

- Ad es. se estendiamo la nostra applicazione per consentire a utenti autorizzati di modificare le schede dei dinosauri via Web *o se vogliamo fornire pagine iniziali di accesso differenziate sulla base di storia precedente o livello di expertise utente*, anche la nostra applicazione cessa di poter essere stateless

36

36

Diversi tipi di stato

Parlando di applicazioni Web è possibile classificare lo stato in modo più preciso:

- **Stato di esecuzione** (insieme dei dati parziali per una elaborazione): rappresenta un avanzamento in una esecuzione; *per sua natura è uno stato volatile*, può essere mantenuto in memoria lato server come stato di uno o più oggetti
- **Stato di sessione** (insieme dei dati che caratterizzano una interazione con uno specifico utente): la sessione viene gestita di solito in modo unificato attraverso l'uso di istanze di oggetti specifici (supporto a *oggetti sessione*)
- **Stato informativo persistente** (ad esempio gli ordini inseriti da un sistema di eCommerce): viene normalmente mantenuto in una *struttura persistente come un database*

37

37

Il concetto di sessione

La **sessione** rappresenta lo stato associato ad una sequenza di pagine visualizzate da un utente:

- Contiene tutte le informazioni necessarie durante l'esecuzione
 - Informazioni di sistema: IP di provenienza, lista delle pagine visualizzate, ...
 - Informazioni di natura applicativa: nome e cognome, username, quanti e quali prodotti ha inserito nel carrello per un acquisto, ...
- Lo **scope di sessione** è dato da:
 - **Tempo di vita** della interazione utente (*lifespan*)
 - **Accessibilità**: usualmente concesso alla richiesta corrente e a tutte le richieste successive provenienti dallo stesso processo browser

38

38

Sessione come base per il concetto di conversazione

- La **conversazione** rappresenta una sequenza di pagine di senso compiuto (ad esempio l'insieme delle pagine necessarie per comperare un prodotto)
- È univocamente definita dall'insieme delle pagine che la compongono e dall'insieme delle interfacce di input/output per la comunicazione tra le pagine (**flusso della conversazione**)

39

39

Esempio di conversazione: acquisto online

1. L'utente inserisce username e password: **inizio della conversazione**
 - Server riceve i dati e li verifica con i dati presenti nel DB dei registrati: viene **creata la sessione**
2. L'utente sfoglia il catalogo alla ricerca di un prodotto
 - Server lo riconosce attraverso i *dati di sessione*
3. L'utente trova il prodotto e lo mette nel carrello
 - *Sessione viene aggiornata* con informazioni del prodotto
4. L'utente compila i dati di consegna
5. L'utente provvede al pagamento, **fine della conversazione di acquisto**
 - L'ordine viene salvato nel DB (stato persistente)
 - La sessione è ancora attiva e l'utente può fare un altro acquisto o uscire dal sito

40

40

Tecniche per gestire lo stato

Lo stato di sessione deve presentare i seguenti requisiti:

- Deve essere condiviso da Client e Server
- È associato a una o più conversazioni effettuate da un singolo utente
- Ogni utente possiede il suo singolo stato

Ci sono due tecniche di base per gestire lo stato, *non necessariamente alternative ma integrabili*:

- Utilizzo del meccanismo *dei cookie (storage lato cliente)*
- Gestione di uno *stato sul server per ogni utente collegato (sessione server-side)*

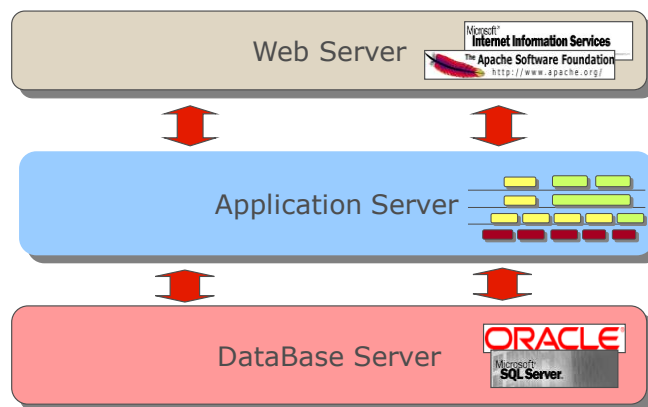
La gestione della sessione è uno dei supporti orizzontali messi a disposizione da un **application server**

41

41

Architettura frequente nei sistemi Web

Architettura a 3 tier, che può collassare a due in assenza di application server (sempre più raro al giorno d'oggi)



42

42

Distribuzione dei servizi

- La struttura a 3 tier rispecchia i 3 principali servizi che realizzano un sistema Web
- Questi 3 servizi possono risiedere sullo stesso HW oppure essere divisi su *macchine separate* (**distribuzione verticale dell'architettura**)
 - Non necessita di nessun accorgimento specifico
 - Viene realizzata essenzialmente per motivi di performance, soprattutto quando si separa il livello applicativo da quello database
- Non prevede replicazione, non è quindi utile per risolvere problemi di fault tolerance

43

43

Distribuzione dei servizi

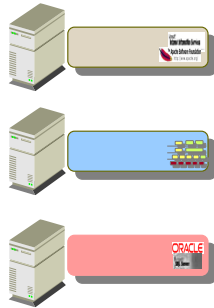
- Orizzontalmente ad ogni livello è possibile replicare il servizio su diverse macchine
- Si parla in questo caso di **distribuzione orizzontale**
 - Necessità di importanti “accorgimenti” strettamente dipendenti dalla tecnologia d'uso
 - Quali vi vengono in mente?
- Essendo una distribuzione per replicazione è possibile implementare *politiche per la gestione della fault tolerance e anche del bilanciamento di carico* a fine di maggiori performance

44

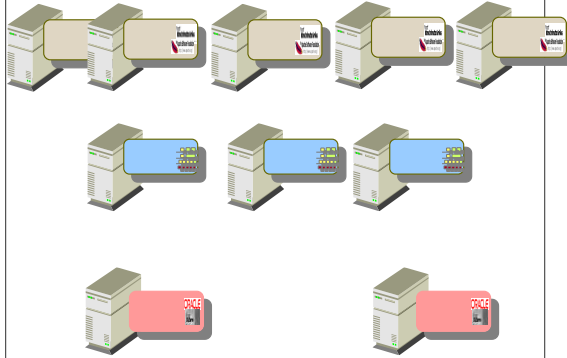
44

Distribuzione verticale e orizzontale

Distribuzione Verticale



Distribuzione Orizzontale

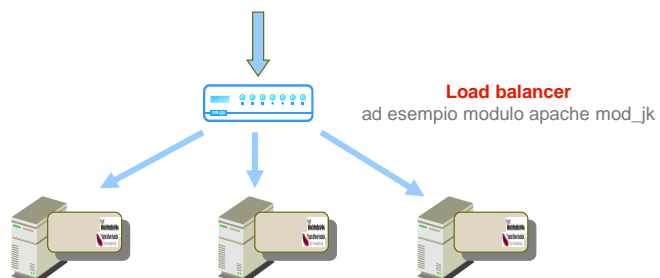


45

45

Replicazione: Web server

- Web server è stateless per la natura del protocollo HTTP; per questo, molto facile da replicare
- Il fatto che IP è embedded in URL può essere gestito attraverso diverse soluzioni sia hw che sw
- Si possono applicare politiche di **load balancing** con diverse euristiche usando dispositivi appositi



46

46

Replicazione: applicazione

- *Stato di sessione prevalentemente*
- Può accadere però che application server utilizzi oggetti o componenti con stato per motivi di performance (cache) o altre necessità specifiche
- Alcuni framework disponibili sul mercato permettono **replicazione attraverso tecniche di clustering** (ne daremo cenni nella seconda parte del corso); **altri framework non sono in grado di replicare orizzontalmente**
- Se si mantiene lo stato concentrato all'interno della sessione e la sessione viene gestita interamente attraverso cookie, è possibile realizzare un framework applicativo completamente stateless lato server, ottenendo così realizzazione più semplice e primitiva di configurazione completam. replicabile in modo orizzontale. **Con quali limiti?** È quello che vi aspettate accada in sistemi commerciali?

47

47

Replicazione: database

- Il database server è (normalmente) un server stateful. **Perché? Con quali problemi conseguenti?**
- **La replicazione** è molto delicata perché deve mantenere il principio di atomicità delle transazioni
- I database commerciali, come Oracle e Microsoft SQL Server prevedono delle configurazioni di **clustering** (forse, lo vedrete meglio all'interno del corso di Sistemi Distribuiti M...) in grado di gestire in modo trasparente un numero variabile di CPU e macchine distinte
 - Comunque in *numero basso (qualche unità)*

48

48