Sistemi Distribuiti

DaveRhapsody

11 Marzo 2020

Indice

Capitolo 1

Introduzione

Un sistema distribuito è un insieme di componenti hardware e software che comunicano tramite scambi di messaggi in rete. Il concetto di componente è fondamentale, può essere per l'appunto sia hardware che software.

Nel dettaglio: E' un sistema di componenti **autonome**, nel senso che le componenti sono indipendenti tra loro MA concorrono tutte allo stesso scopo. Per apparire come unico componente occorre generare una sorta di collaborazione **SENZA** memoria condivisa.

Autonomia: Ognuno svolge il proprio compito in modo indipendente (sia in termini di tempo che dati) da ogni altro. Pertanto occorre che le componenti siano **SINCRONIZZATE** (Reti e Sistemi insegna ->). Questo insieme di nodi appaiono all'utente finale come un unico sistema **coerente** senza che lui sappia dove vengono processati (e nemmeno come) i dati. Il sistema è un blocco unico agli occhi dell'utente MA obv no.

Concetto di trasparenza: Il livello di trasparenza lo si decide in fase di progettazione del sistema, fondamentalmente è il livello tale per cui l'utente sappia come vengono processati i dati, e cosa e come viene eseguito dal sistema. (Un sistema trasparente, è un sistema che ti fa vedere vita, morte, miracoli, luci, coriandoli, sassi e Babbo Natale).

1.1 Caratteristiche fondamentali

- Non c'è memoria condivisa
- L'esecuzione è concorrente (allo stesso istante)
- Non ci sono stati del processo o della memoria, ogni nodo è a sè un processo!
- NON C'E' UN CLOCK GLOBALE, niente scheduling. controllo (globale), si fa tutto per scambio di messaggi
- Non esiste un fallimento globale ma un fallimento della singola componente.

  Sistemi Realtime: Noi non utilizziamo questo genere di dispositivi, studiamo e prendiamo in esame dispositivi best-effort

- Architetture software

1.2 Architetture software

E' la struttura del sistema, di come sono organizzate le componenti, quali sono i protocolli, le interfacce, etc.

- Architetture a Livelli (tier)
- Architetture a oggetti
- Architetture centrate sui dati
- Architetture event-based (ajax)

1.2.1 Stratificazione

L'idea è formare degli strati di complessità, fondamentalmente nulla vieta di fare tutto a livello application, lì si impreca per davvero, ma soprattutto non ci sarebbe una specializzazione. Il motto è "Fai una cosa, ma falla bene".

1.3 Tipi di sistemi operativi

In base al tipo ho un livello diverso di trasparenza

- DOS: Distributed OS
- NOS: Network OS: L'OS ti dà delle librerie e supporti per supportare delle applicazioni MA non nasconde le comunicazioni tra le varie applicazioni.
- Middleware: Hai un insieme di sistemi di rete che forniscono primitive per sostenere comunicazione tra sistemi E costruirci delle logiche che consentano di vedere questo come unico sistema (Sì, il middleware è a sua volta un'applicazione distribuita)

1.4 Servizi del middleware

- Naming
- Accesso trasparente
- Persisenza: avviene una memorizzazione di dati persistente
- Transazioni distribuite: Va garantita la consistenza dei dati
- Sicurezza dei dati: integrità computazionali

Riporto il riassunto presente nelle slides

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Capitolo 2

Modello Client-Server

E' un'interazione basata su richiesta-risposta tra una componente e l'altra.

Funzionamento:

1. Client invia la request
2. Server la riceve, client aspetta
3. Server elabora
4. Server risponde alla richiesta

Quando client aspetta, quest'ultimo è in standby, in attesa, fermo, inchiodato. Il server si attiva ad ogni richiesta che arriva. Richiesta e risposta hanno un tempo ovviamente proprio, che deriva dalla rete di riferimento, chiaramente se si fa tutto sulla stessa macchina si parla di microsecondi.

Osservazione: Un dispositivo è sia client che server, può essere sia uno che l'altro.

Tipi di accesso:

- Accesso a server multipli (nel senso che sono duplicati i server)
- Accesso via proxy

2.1 Problemi dei Sistemi Distribuiti

- Identificazione della controparte: Identifico la risorsa, tipo con address etc.
- Accesso alla controparte: Dove accedo? Chi è il mio access point
- Comunicazione 1: Definisco il formato dei messaggi scambiati
- Comunicazione 2: Capisco il contenuto del messaggio in seguito all'estrazione

  Esempio: I tipi per i dati, che specificano un insieme di valori ed operazioni fattibili su un determinato dato.

2.2 Trasparenza

Significa nascondere all'utente **come** viene ottenuto un determinato risultato. E' per intenderci ciò che salva coloro che programmano tutto nel main. Come si fa?

- **Naming**: non mi connetto per indirizzo IP ma per indirizzo web, link.
- **Accesso trasparente**: Come accedo in maniera trasparente? Anche qui nomi simbolici, qualcosa che non mi faccia capire la locazione della controparte
- **Location Trasparency**: Hai una stampante a casa e devi stampare una foto di gattini? Ti serve sapere dove si trova, non puoi cercare a caso su google, quindi devi sapere fisicamente dove hai la stampante. Invece se ti serve una calcolatrice online tipo wolfram alpha, digiti l'equazione, la risolve, ma tu non sai una bega di dove è stata risolta.
- **Relocation o trasparenza mobile**: Posso spostare le risorse mentre le uso (Cellulari, dispositivi wifi, ci siamo capiti).
- **Migrazione**: Posso spostare un sito web da un pc del 91 ad un pc del 2020, il servizio quello è, cambierà la velocità (tempi di elaborazione + request + response)
- **Replicazione**: Hai una serie di server identici, tu ti connetti ad uno o l'altro, cambia niente
- **Concorrenza**: In più utenti accediamo allo stesso servizio (spotify)
- **Trasparenza del fallimento**: Ciò che resta dopo un fallimento di una componente deve colmare i danni di una componente morta.
- **Persistenza**: Non mi accorgo di quando un sistema è riavviato o no

In alcuni casi è impossibile nascondere un fail o lo stato di una applicazione, se ti crasha Word, vedi che ti è crashato Word, quindi in qualche modo va comunicato all'utente "Ueh, fa schifo Uord, usa OpeNoFFiSsszsxxs". Ma tra l'altro non sempre è utile nascondere, prendete l'esempio di prima della stampante.

2.3 Separazione tra interfaccia e realizzazione

Costruisco un'astrazione logica che nasconde i dettagli implementativi di ciò che sta diestro. Ogni componente pubblica il What cioè COSA fa, ma non ti spara fuori anche l'HOW, cioè COME lo fa, è il concetto di information hiding di Java.

Esempio: Gestione delle stringhe nei vari linguaggi. Una stringa è una concatenazione di caratteri, la gestione di append, copy, concat etc. sono how, il risultato finale è il what

2.4 Politiche versus mechanism

Un Sistema distribuito è composto da.. Va beh avete capito. Progressivamente si va verso un unico enorme (utopico) sistema complesso organizzato con delle policy stabilite. Le politiche banalmente sono una serie di regole ad esempio prendete il context switch di Unix.

- Il CS è un meccanismo
- Il Round Robin invece è la policy di come è gestito un comportamento

2.5 Concetto di protocollo

Un protocollo è un insieme di regole che definiscono un formato, l'ordine, payload, operazioni da compiere alla ricezione od all'invio di un messaggio.

Capitolo 3

Le Socket

Cos'è? Una socket è un'interfaccia tra il sistema operativo ed un processo su di esso eseguito, quindi un programma in stato di esecuzione. **Come?** Fondamentalmente dati due processi, la socket è colei in grado di metterli in comunicazione inviando o leggendo dati.

3.1 Aspetti critici

3.1.1 Gestione del ciclo di vita di client e server

Quando si attiva e/o disattiva uno dei due, chi la gestisce è il middleware, oppure può essere manuale

3.1.2 Identificazione ed accesso al server

Un client per accedere ad un server deve conoscerne l'indirizzo ed una serie di informazioni che consentano di raggiungerlo.

Come fa un client a conoscere l'indirizzo server? Ha diverse alternative

- Inserisce nel codice del client l'indirizzo del server espresso come costante (Il client di un servizio bancario)
- Si chiede all'utente di inserire l'indirizzo (E' quello tipico di browser e simili)
- Si può utilizzare un **name server** o si attinge da un repository da cui il client può acquisire informazioni necessarie
- Si adotta un protocollo differente per individuare il server (ad esempio il DHCP)

3.1.3 Comunicazione client server

Quali sono le primitive disponibili? Quali sono le modalità di comunicazione?

3.1.4 Ripartizione dei compiti client e server

Dipendentemente dal tipo di applicazione si decide chi fa cosa, quali compiti spettano al server, quali al client. Come si decide? Si deve fare un'analisi dei rischi in fatto di sicurezza, prestazioni etc.

3.2 Soluzioni agli aspetti critici

1. Si identifica la controparte, tramite il naming, ogni host avrà un nome, si fa riferimento ad un indirizzo ed una porta
2. Per accedere alla controparte servirà accedere all'indirizzo:porta, usando direttamente l'indirizzo IP (access point)
3. Comunicazione 1: (Protocolli) Stream di byte
4. Comunicazione 2: (Sintassi e semantica) Metodo di interpretazione dei flussi di byte

Il livello di trasparenza è basso poichè è il programmatore che deve preoccuparsi di queste problematiche

3.3 Comunicazione via socket

Le comunicazioni in TCP/IP avvengono tramite flussi di byte in seguito ad una connessione esplicita con systemcall read/write

Perchè? Perchè sono due funzioni sospensive, ovvero bloccano il processo finchè non è raggiunto il completamento dell'operazione e soprattutto viene utilizzato un buffer per garantire flessibilità

Come funziona?

1. Il server richiede di aprire un canale in ascolto (Socket bind)
2. Il client con una socket crea un canale analogo, senza una bind, ma con stesso effetto finale
3. Il client invia la richiesta, il cui formato è stabilito dal livello di trasporto
4. Il server crea un secondo canale di comunicazione che sarebbe l'accept, quindi avviene l'handshake, poi si procede con il trasferimento

Non si comunica sulla stessa socket poichè da una socket o si entra o si esce, non si possono fare entrambe le cose.

Capitolo 4

Tipi di comunicazione

Ci sono fondamentalmente diversi tipi di comunicazione. Il primo tipo è la comunicazione orientata ai messaggi. Per capire meglio come funzioni quest'ultima si fa riferimento all'architettura del Web, ma prima definiamo la comunicazione orientata ai messaggi.

4.1 Comunicazione orientata ai messaggi

Come accennato prima, questo tipo di comunicazione prevede lo scambio di messaggi, generalmente uno richiesta e uno risposta, ed è tutto gestito e controllato da dei protocolli.

4.1.1 Protocolli

Un protocollo è un insieme di regole atte a stabilire formato, ordine di invio e ricezione, dispositivi, tipi per i dati ed azioni da eseguire in invio e ricezione di un messaggio

Alcuni esempi di protocolli: HTTP, SFTP, SMTP, FTP, verranno definiti in seguito.

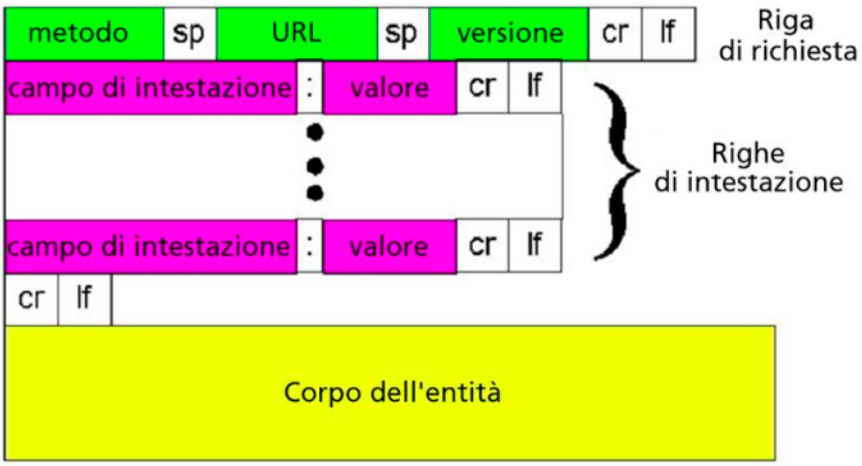
4.1.2 Protocollo HTTP

Il web (dopo si vedrà come) lavora prevalentemente con HTTP (hypertext transfer protocol), quindi tramite scambio di HTTP requests e responses tra client e server, è fondamentalmente il protocollo in grado di trasferire dell'ipertesto.

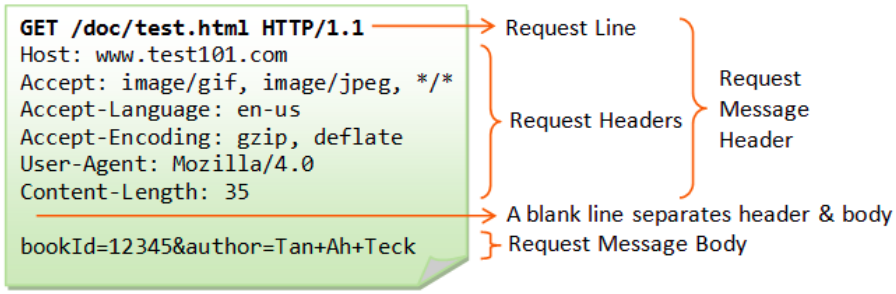
Cosa c'è alla base di HTTP? Livello transport: TCP, nello specifico instanzia una socket sulla porta 80, il server accetta la connessione (Handshake a 3 vie) e si effettuano gli scambi

E' un protocollo stateless: Il server non si ricorda delle richieste precedenti, quindi una richiesta deve avere tutte le informazioni richieste per l'esecuzione. Ci sono altri protocolli che mantengono però l'informazione.

Formato dei messaggi HTTP: Request e response hanno la stessa struttura:



Esempio di messaggio HTTP request:



4.1.3 Metodi per effettuare HTTP requests

Sono fondamentalmente 3:

Metodo GET: Restituisce una rappresentazione di una risorsa, include un eventuale input in coda alla URL della risorsa, l'esecuzione non ha effetti sul server ed è gestible da una cache del client

Specificazione: Una get può essere condizionale, per evitare di inviare roba che il client ha già nella cache. Ad esempio ci puoi mettere un if-modified-since: data che banalmente non trasferisce nulla se l'ultima modifica è entro una certa data.

E la risposta? Il server dà risposta vuota se l'oggetto è aggiornato. HTTP/1.0 304 Not Modified

Uso tipico: Ottenere dati in formato di pagine html e immagini

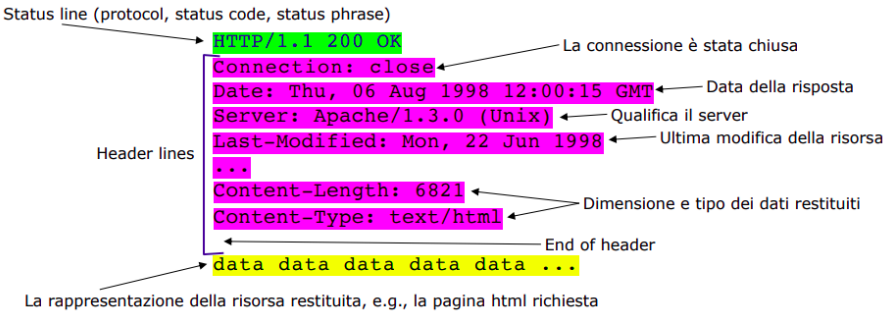
Metodo POST: Comunica dei dati che vengono elaborati lato SERVER o crea una risorsa subordinatta all'URL, inoltre non è idempotente, ogni esecuzione ha un effetto diverso, e la risposta non è gestibile con una cache lato client. E' usata per gestire i FORM o modificare i DB

Metodo HEAD: Molto simile alla GET ma restituisce solo l'HEAD della pagina web, si usa spesso in fase di debug.

Ecco una tabella riassuntiva dei metodi HTTP:

		cache	safe	idempotent
OPTIONS	represents a request for information about the communication options available on the request/response chain identified by the Request-URI			✓
GET	means retrieve whatever information (in the form of an entity) is identified by the Request-URI	✓	✓	
HEAD	identical to GET except that the server MUST NOT return a message-body in the response	✓	✓	
POST	is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line			
PUT	requests that the enclosed entity be stored under the supplied Request-URI			✓
DELETE	requests that the origin server delete the resource identified by the Request-URI			✓
TRACE	is used to invoke a remote, application-layer loop- back of the request message			✓

Formato di una HTTP response:



4.1.4 I tipi MIME

Nella parte del web si vedrà una specificazione di questi, fodamentalmente MIME significa Multipurpose Internet Mail Extension, qualifica i dati via Internet (in HTTP, qualifica il tipo dei dati del body)

Quali sono questi tipi? Testo, immagini, video, audio, dati che devono essere processati da un'applicazione prima di esser visibili

4.1.5 I cookie

Un cookie serve per legare più richieste in modo da associare un ID alla conversazione. Il client manda al server un cookie, che è un numerello, ed il server saprà chi è che parla ogni volta che riceve quel cookie per gli accessi positivi.

Come vien controllato? Il server autentica ogni cookie, traccia le preferenze dell'utente e la sessione di lavoro (Sì, esatto, quella roba dei cookie che si legge in ogni sito serve a questo)

Come controllo l'accesso ai documenti lato server? Consideriamo che HTTP è stateless, il client deve autenticare ogni richiesta, perciò molto semplicemente il client manda login e password nell'header del messaggio, e per evitarsi che l'utente li digiti ogni volta, se li salva. Not bad!

4.2 Comunicazione orientata allo stream

Questa cosa è stata menzionata nella parte relativa al livello di trasporto del modello TCP/IP, fondamentalmente è la differenza tra TCP e UDP. TCP numera ogni elemento della sequenza, garantisce la correttezza dell'informazione, l'ordine, e l'integrità. Invece UDP di questo non si cura.

E al lato applicativo? L'applicazione invia i messaggi come stream di byte al livello di trasporto, e poi leggerà i byte dal trasporto ricostruendo i messaggi. Ma non si cura di ciò che c'è sotto

4.3 Architettura del web

Il web supporta l'interazione tra client e server mediante HTTP, e a seconda del fatto che un host sia client o server cambierà il tipo di messaggio che invia (o riceve).

- Se si tratta di un **Server** riceverà delle HTTP **Requests**
- Se si tratta di un **Client** riceverà delle HTTP **Response**

Client e server sono rispettivamente rappresentati da:

- Client: un Browser, od un User-Agent
- Server: un web server o HTTP server

4.4 Tipi di comunicazione

Esistono diversi tipi di comunicazione:

1. Comunicazione sincrona o asincrona
2. Comunicazione transiente (se il destinatario non c'è, i dati sono scartati)
3. Comunicazione persistente: Il middleware memorizza i dati fino alla consegna del messaggio, non serve che i processi siano in esecuzione prima e dopo l'invio.

4.4.1 Comunicazione persistente

Un messaggio persistente può esser sia sincrono che asincrono, la differenza nel fatto che nel caso dell'asincrono, dati A e B che voglion comunicare:

1. A invia un messaggio e continua a fare il suo
2. B si sveglia ad un certo punto, e riceve il messaggio

Mentre nel caso il messaggio è sincrono

1. A invia un messaggio a B e si ferma ad aspettare
2. B non è in esecuzione ma il messaggio verrà salvato in un buffer che leggerà al risveglio

3. A riceve l'ok dell'inserimento del dato in quel buffer, e torna a fare il suo
4. B si sveglia, e riceve il messaggio

Nel caso di una comunicazione asincrona transiente:

1. A manda il messaggio e fa il suo
2. B riceverà solo se è in esecuzione

Nel caso invece la comunicazione è sincrona A invia e aspetta che b riceva e risponda. Invece per le comunicazioni transienti delivery-based si ha che:

1. A invia una richiesta e aspetta
2. B si sta facendo i cazzi suoi da acceso, riceve, risponde con comodo
3. A riparte da quando riceve l'accepted, e poi B processa la sua richiesta

Invece nel caso sia una response-based..

1. A invia il messaggio, e non solo aspetta che B risponda, ma aspetta che B finisca di processare, producendo un Sent-request delay.

4.4.2 Web browser

Considerando il lato client si ha questo software che consente di visualizzare le pagine web, è detto **User-Agent**, fondamentalmente ciò che fa un browser è **interpretare** un codice di mark up (HTML, ad esempio), che sarebbe una pagina web, e successivamente visualizzarlo (in forma di ipertesto)

Mh.. Manca qualcosa: Assieme all'HTML (che delinea la grafica) un browser consente (soprattutto se $browser \neq Internet - Explorer$) molta più roba, dalla multimedialità, ai feed RSS, ai file XML, Json etc.

4.4.3 La pagina web

Come accennato nella sottosezione precedente, si hanno delle vere e proprie pagine web, che fondamentalmente sono dei file, dei documenti, costituiti da un certo insieme di oggetti, o **risorse**.

Definizione di risorsa: Una risorsa è una **sequenza di dati identificata da un URL**, che per intenderci è un indirizzo univoco. Questo insieme di dati risiede su **memoria di massa** e quindi è per semplificarci la vita un **file**.

Okay, che file abbiamo? La maggior parte delle pagine web ha almeno una riga di HTML (Hypertext Markup Language), anche solo per definire i contenuti testuali e qualche elemento grafico

E il web server? Il web server è l'applicativo che ha il compito di gestire quei famosi file, e soprattutto che li rende disponibili ai client (Se c'è un mondo dietro ai browser, per fare a gara tra quale è il migliore, c'è anche lato server. Sì.)

4.4.4 L'URL

Un URL (Uniform Resource Locator) identifica un oggetto nella rete, ma non è questo il lato più importante

Un URL specifica il protocollo con cui si possono interpretare i dati .

Come è composto? E' composto da 5 componenti principali:

1. Nome del protocollo (sftp, http, https, ftp, smtp etc.)
2. Indirizzo dell'host (IP address)
3. Porta di destinazione
4. Percorso file nell'host
5. Risorsa

Esempio: accedi in ftp a 192.168.50.5 al percorso /media/hdd e prendi il file prova.txt ftp://192.168.50.5:21/media/hdd/prova.txt

4.4.5 Ipertesto

Per tre volte si è menzionato l'**ipertesto**, ma cosa sarebbe? Fondamentalmente è un insieme di testi o pagine leggiili tramite un'interfaccia elettronica. Funzionano tramite hyperlink, o anche solo link.

E che fanno i link? Costituiscono una rete raggiata, o incrociata di informazioni organizzate su citeri paritetici o gerarchici (I menù, per capirci)

4.4.6 Linguaggi del web

I dati testuali si esprimono con linguaggi standard (HTML, XML e JSON), rispettivamente descrivono: Struttura contenuti (HTML) e struttura dei dati (JSON, XML)

E i dati non testuali? Esiste una tecnica di encoding detta MIME che consente di definire il formato dei contenuti non testuali (video, audio, etc.)

Ma gli script? Lasciati in disparte fin dal principio, questi sono essenziali, consentono la creazione di pagine web **dinamiche**. Alcuni dei più famosi: VBScript, PHP, JavaScript, Adobe Flash etc.

Capitolo 5

HTML, DOM, CSS, JAVASCRIPT

Sono tutti linguaggi, sebbene non tutti dello stesso tipo. Di fatto Javascript è un linguaggio di programmazione (In questo caso interpretato) mentre per quanto riguarda HTML e CSS si fa riferimento a linguaggi di Markup

5.1 Linguaggi di markup

Cosa sono? In modo semplificativo, sono linguaggi che dicono all'interprete come rappresentare graficamente un foglio, o una pagina (in questo caso web).

Solo per pagine web? No, questo documento è stato scritto in LaTeX, che è un linguaggio di MarkUp per documenti PDF, però non è solo questa la differenza. Infatti LaTeX è compilato, mentre HTML, Markdown e simili sono tutti interpretati.

Chi è l'interprete di HTML+CSS? Qualsiasi browser, addirittura anche Internet Explorer (Con un quarto d'ora di ritardo, ma c'è).

5.1.1 HTML

Hypertext Markup Language, per l'appunto, linguaggio di markup per strutturare contenuti web (L'ipertesto). Quando si è di fronte ad una pagina web, 9/10 è scritta, o contiene, del codice in HTML

5.1.2 CSS

Cascading Style Sheets, è un linguaggio di supporto all'HTML che invece serve per dare uno stile di presentazione ai contenuti, quindi fondamentalmente è tipo una libreria di quelle che si importa in Java o C. Da solo non serve a niente, ma integrandolo con HTML, porta a pagine carine

Diversi stylesheet: Ne esistono diversi, per l'appunto

- L'autore della pagina in genere ne specifica uno, o anche più di uno
- Il browser ne ha uno solo, o un vero css, oppure uno simulato nel codice del browser

- Il lettore, l'utente del browser, ne può definire uno personale per modificare la propria esperienza d'uso.

5.1.3 DOM

Document Object Model, è un'interfaccia neutrale rispetto al linguaggio di programmazione, è fondamentalmente quello che consente ad una pagina web di essere dinamica ed avere del codice sotto che funziona. Form, elenchi, quella roba lì.

Precisazioni: Usare HTML e CSS serve per fare in modo che contenitore e contenuto siano indipendenti l'un dall'altro, la separazione dei concetti infatti è ormai d'obbligo in informatica, PERO' lo stile può avere un coupling, un legame con il contenuto.

Il browser infatti non è un lettore di pagine scritte in HTML e morta lì, ma è un ambiente di sviluppo che fornisce un numero enorme di funzionalità. Se premete su una qualunque pagina F12 capirete meglio il concetto.

Capitolo 6

Servlet e Applicazioni Web

Prima di capire cos'è una servlet serve dare una veloce spiegazione di come funziona un'applicazione web.

Come spiegato in qualche precedente capitolo, tutto è basato su un'architettura client-server, quindi ci sarà un Web Client (Browser) ed un web server (Apache o simili).

Come si passa dal Web alle applicazioni? Partendo dal fato che il web supporta l'interazione tra client e server via HTTP, il server avrà un programma che si chiama Application Server (E' UN SOFTWARE, NON HARDWARE) che è caratterizzato dal protocollo di interazione con il Web Server stesso. L'interazione con il client è infatti tutta in HTTP.

Cosa accade lato server? Ogni calcolo, ogni computazione, viene svolta dal server, e può avvenire sia da scriptini che da programmi compilati. Nel caso di software compilati ciò che accade è che il server invoca l'eseguibile (Partendo dalla richiesta del client).

Come vengono eseguiti gli script? Il Web Server ha un motore in grado di intepretare il linguaggio di scripting di riferimento, quello richiesto, per intenderci

6.1 Sigle da sapere

1. URL: Uniform resource locator, definisce un naming globale, un modo per tracciare le cose

2. CGI: Content Gateway Interface, permette al server di attivare un programma, passargli le richieste e i parametri dal client, e recuperare una risposta

   Che vantaggi ci sono?

   - Ti devi programmare solo le logiche
   - Hai un modello di applicazioni conformi al modello del linguaggio
   - Semplice, riproducibile ovunque, manutenibile in modo agevole

6.2 Servlet

Finalmente si arriva alle Servlet. Che sono? Sono applicazioni Java che si trovano sul server. E' un componente che il server gestisce in modo automatico da un engine, oppure da un container. Dalle slide c'era qualche esempio di come si costruiscono e usano, di importante c'è solo da tenere a mente che:

(a) Sono componenti gestite da un container o un engine

(b) Hanno un'interfaccia che definisce i metodi ridefinibili (Per esigenze di standardizzazione)

(c) Risiedono in memoria, mantenendo uno stato e possono interagire con altre Servlet

(d) HTTP non consente di mantenere un'informazione da un messaggio ai successivi, quindi questo lo fa la Servlet con Cookies e HTTPSessions

6.3 Cosa implementa l'interfaccia javax.servlet.Servlet

`void init(ServletConfig config)`
Inizializza la servlet

`void destroy()`
Chiamata quando la servlet termina (es: per chiudere un file o una connessione con un database)

`void service(ServletRequest request, ServletResponse response)`
Invocato per gestire le richieste dei client

`ServletConfig getServletConfig()`
Restituisce i parametri di inizializzazione e il ServletContext che da accesso all'ambiente

`String getServletInfo()`
Restituisce informazioni tipo autore e versione

Sono presenti comunque due classi astratte che implementano questi metodi in modo che non facciano nulla:

1. javax.servlet.GenericServlet
   Praticamente implementa **service** che invoca i metodi per gestire le richieste dal web:
   - doX con X che è un metodo HTTP (doGet, doPost)
   - Parametri: HTTPServletRequest e HTTPServletResponse
   - Eccezioni: ServletException e IOException
2. javax.servlet.http.HTTPServlet

Anche i parametri sono stati adattati al protocollo HTTP, cioè consentono di ricevere(inviare) messaggi HTTP leggendo(scrivendo) i dati nell'head e nel body di un messaggio.

1. HTTPServletRequest
   - Viene passato un oggetto da **service**
   - Contiene la richiesta di un client
   - E' un'estensione di ServletRequest

2. HTTPServletResponse
    - Viene passato un oggetto anche qua da **service**
    - condiene però la risposta PER il client
    - Estende (manco a dirlo) ServletResponse

6.3.1 Metodi per richieste e risposte

- I metodi principali per le richieste
  `String getParameter(String name)`
  Restituisce il valore dell'argomento name (valore singolo)
  `Enumeration getParameterNames()`
  Restituisce l'elenco dei nomi degli argomenti
  `String[] getParametersValues(String name)`
  Restituisce i valori dell'argomento name (valore multiplo)

- I metodi principali per le risposte
  `void setContentType(String type)`
  Specifica il tipo MIME della risposta per dire al browser come visualizzare la risposta
  Es: "text/html" dice che è html
  `ServletOutputStream getOutputStream()`
  Restituisce lo stream di byte per scrivere la risposta
  `PrintWriter getWriter()`
  Restituisce lo stream di caratteri per scrivere la risposta

Poi ci sono altri metodi legati ai cookies:

Altri metodi

`Cookie[] getCockies()`
Restituisce i cookies del server sul client

`void addCookie(Cookie cookie)`
Aggiunge un cookie nell'intestazione della risposta

`HTTPSession getSession(boolean create)`
Una HTTPSession dentifica il client.
Viene creata se create=true

6.4 Come si esegue una servlet

Tutto parte da un HTTPServer e un Servlet container che per l'appunto contiene la Servlet (In memoria, come specificato nella sezione precedente). Semplicemente viene creata un'istanza della servlet condivisa da tutti i client, ed ogni richiesta genera un nuovo Thread che esegue doX in base a cosa è stato richiesto.

6.4.1 Ciclo di vita

1. Dal container si crea la Servlet
2. Si invoca il metodo init() per inizializzazioni specifiche
3. Una servlet viene distrutta all'occorrenza di uno o due eventi:

(a) Quando non ci sono servizi in esecuzione

(b) Quando è scaduto un timeout predefinito

4. Si invoca il metodo destroy() che termina la servlet e ne dealloca le risorse.

Principali problemi:

(a) Container e richieste devono sincronizzarsi sulla terminazione poichè alla scadenza di un timeout magari la service() è ancora in esecuzione e quindi

i. Bisogna tener traccia dei Thread in esecuzione

ii. Bisogna progettare il metodo destroy() in modo da notificare gli shutdown ed attendere il completamento di service()

iii. Bisogna progettare metodi lunghi in modo che si verifichi periodicamente se è in corso uno shutdown

6.5 JSP: Java Server Pages

E' una tecnologia per creare applicazioni web. Specifica l'interazione tra un container ed un insieme di pagine che l'utente avrà davanti.

Perchè? Perchè rispetto alle servlet, facilitano la separazione tra logica applicativa e presentazione

In pratica servono a separare la parte dinamica dal codice statico HTML infatti il codice si racchiude tra "<%" e "%>". La pagina viene convertita automaticametne in una servlet java la prima volta che vien richiesta

6.5.1 Ciclo di vita di una applicazione JSP

1. Il client manda una richiesta al WebServer

2. Il webserver contatta il Servlet Container

3. Dal Servlet Container si richiama una JSP che compila la Servlet con il JSP compiler, e infine, viene eseguita la Servlet dal Servlet container

6.5.2 Elementi di una JSP

- Template text
  - Le parti statiche della pagina
- Commenti

```
<%-- questo è un commento -->
```

- Direttive

```
<%@ direttiva ... di compilazione %>
```

- Azioni

```
<jsp:XXX attributes> body </jsp:XXX>
```

- Elementi di scripting
  - Istruzioni nel linguaggio specificato nelle direttive
  - Sono di tre tipi: scriplet, declaration, expression

- page
  - Liste di attributi/valore
  - Valgono per la pagina in cui sono inseriti

```
<%@ page import="java.util.*" buffer="16k" %>
<%@ page import="java.math.*, java.util.*" %>
<%@ page session="false" %>
```

- include
  - Include in compilazione pagine HTML o JSP

```
<%@ include file="copyright.html" %>
```

- taglib
  - Dichiara tag definiti dall'utente implementando opportune classi

```
<%@ taglib uri="TableTagLibrary" prefix="table" %>
<table:loop> ... </table:loop>
```

6.5.3 Azioni di una JSP

- forward
  - determina l'invio della richiesta corrente, eventualmente aggiornata con ulteriori parametri, all'URL indicata

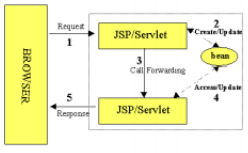
```
<jsp:forward page="login.jsp" >
  <jsp:param name="username" value="user" />
  <jsp:param name="password" value="pass" />
</jsp:forward>
```

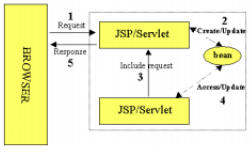
- include
  - invia dinamicamente la richiesta ad una data URL e ne include il risultato

```
<jsp:include page=»shoppingCart.jsp" />
```

- useBean
  - localizza ed istanzia (se necessario) un javaBean nel contesto specificato
  - Il contesto può essere
    - La pagina, la richiesta, la sessione, l'applicazione

```
<jsp:useBean id="cart" scope="session" class="ShoppingCart" />
```





6.5.4 Elementi di scripting

- Declaration `<%! declaration [declaration] ...%>`
  - Variabili o metodi usati nella pagina
  - Valgono per la durata della servlet

```
<%! int[] v= new int[10]; %>
```

- Expression `<%= expression %>`
  - Una espressione nel linguaggio di scripting che viene valutata e sostituita con il risultato

```
<p>La radice di 2 vale <%= Math.sqrt(2.0) %> </p>
```

- Scriptlet `<% codice %>`
  - Frammenti di codice che controllano la generazione della pagina, valutati alla richiesta

```
<table>
    <% for (int i=0; i< v.length; i++) { %>
        <tr><td> <%= v[i] %> </td></tr>
    <% } %>
</table>
```

  - Le variabili valgono per la singola esecuzione
  - Ciò che viene scritto sullo stream di output sostituisce lo scriptlet

- Linguaggio di script ha lo scopo di
  - interagire con oggetti java
  - gestire le eccezioni java

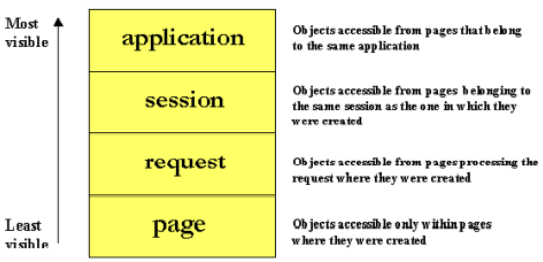
- Utilizza oggetti impliciti: gli elementi delle servlet (sono 9)
  - request
  - response
  - out
  - page
  - pageContext
  - session
  - application
  - config
  - exception

Esempio:

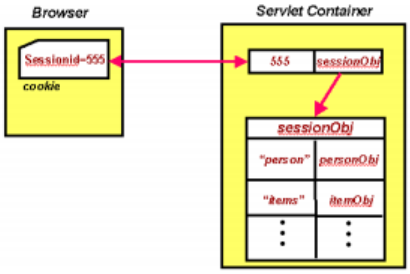
```
Grazie per la scelta di <i> <%= request.getParameter("title") %> </i>
```

6.5.5 Oggetti e loro "scope"

- Gli oggetti possono essere creati
  - implicitamente usando le direttive JSP
  - esplicitamente con le azioni
  - direttamente usando uno script (raro)
- Gli oggetti hanno un attributo che ne definisce lo "scope"



- Accede ad un oggetto HTTPSession



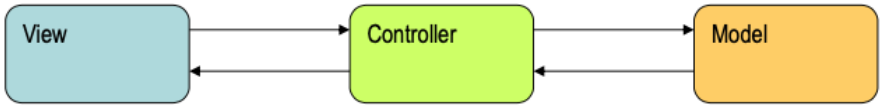
- Esempi
  - Memorizzazione
    ```
    <% Foo foo = new Foo(); session.putValue("foo",foo); %>
    ```
  - Recupero
    ```
    <% Foo myFoo = (Foo) session.getValue("foo"); %>
    ```
  - Esclusione di una pagina dalla sessione
    ```
    <%@ page session="false" %>
    ```

6.6 Pattern Model View Control

Il pattern Model View Controller (Da ora solo MVC) serve per

- Separare i dati ed i metodi per manipolarli (Model)
- Curare l'interfaccia, come si presenta graficamente (View)
- Regolare il coordinamento dell'interazione tra interfaccia (quindi le azioni degli utenti) ed i dati (Controller)

Il tutto secondo questo schema:

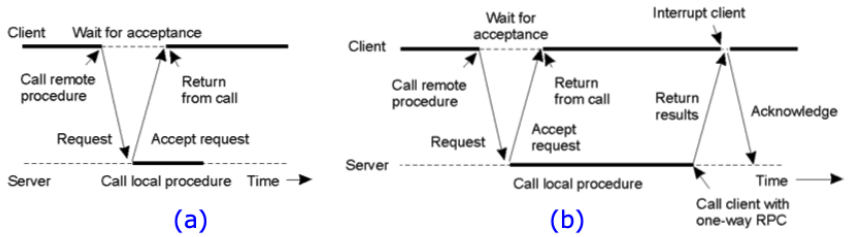


6.7 Remote Procedure Call

Il RPC è un modello che si applica al client-server, come se fosse una vera e propria maschera. Se esiste è perchè ci sono dei vantaggi, ovvero una semantica nota (chiamata di procedura) e perchè si semplifica l'implementazione.

E gli svantaggi? Sono realizzate dal programmatore, quindi tutto è esplicito (basta usare modelli con componenti più complesse). Inoltre sono **Statiche**, scritte direttamente nel codice dei programmi e non c'è concorrenza, quindi son bloccanti. O loro, o nessuno.

Tenendo a mente i modelli di comunicazione tra client e server, questo è ciò che accade con il modello RPC asincrono.



a) interazione tra client e server utilizzando una RPC asincrona, senza risposta
b) interazione tra client e server utilizzando una RPC asincrona, con riposta posticipata utilizzando una seconda RPC asincrona (una call-back)

E come si passa da PC a RPC?

Lato client

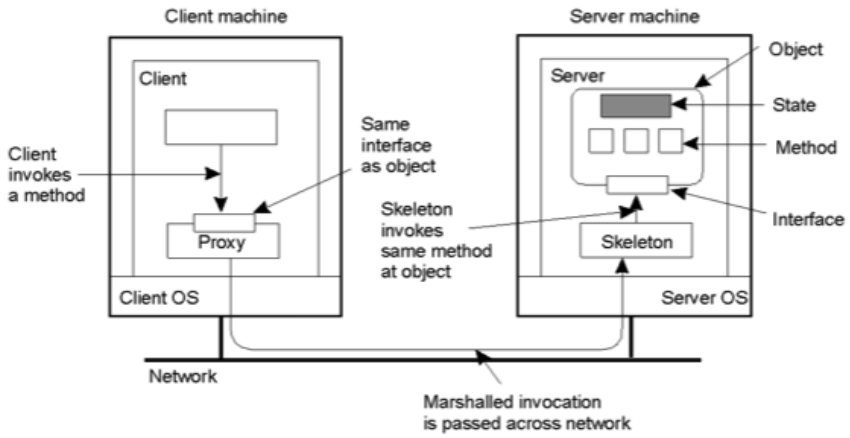
- Programma: chi chiamare?
- Middleware: come interagire il programma cliente?
- Middleware: dove sta la macchina server?
- Middleware: come trattare i parametri?

Lato server

- Middleware: quale procedura invocare?
- Middleware: come passare i parametri?
- Middleware: come trattare il risultato?

6.8 RMI: Distributed Objects

Partiamo con il considerare l'architettura di riferimento



La quale presenta delle sue caratteristiche

- Oggetti
  - Incapsulano i dati (stato) – I valori dei campi o variabili
  - Definiscono operazioni sui dati – I metodi delle classi
  - Definiscono l'accesso – I metodi delle interfacce
- Oggetti a compile-time
  - Definiti attraverso interfacce e classi (Java, C++)
- Oggetti a run-time
  - Accessibili attraverso adapters (wrappers)
- Oggetti persistenti e transienti
- Riferimento agli oggetti remoti

6.8.1 Puntatori e riferimenti

Un puntatore è un tipo per i dati le cui variabili contengono un indirizzo di memoria (Variabile di tipo puntatore, per accorciare una variabile di tipo puntatore è chiamata solo puntatore).

Può essere modificato in qualsiasi istante e non dipende dal tipo per i dati della variabile puntata.

Un riferimento, o anche solo reference in Java, è una variabile che contiene delle infdormazioni logiche (alias) per accedere ad un oggetto, è **immutabile** e dipendente dalla classe, nel senso che la classe dell'oggetto ne definisce il tipo.

Ha senso parlare di puntatore distribuito? No, non avrebbe praticamente senso

Invece una reference distribuita? Sì, presenta un indirizzo della macchina + indirizzo (porta) del server (processo) e soprattutto un **ID** dell'oggetto stesso

6.9 RMI Middleware

Tutto questo preambolo per dire che RMI è un middleware, che estende l'approccio **OO** al distribuito

- Supporta anche l'invocazione di metodi tra oggetti su macchine virtuali distinte
  - Le interfacce sono Java, non in un IDL generico
  - Vengono passati e tornati oggetti Java
  - Le classi vengono caricate dinamicamente
- Si basa sulla portabilità del bytecode e sulla macchina virtuale: quindi è più **sicuro** ed **efficiente** perchè non si deve tradurre nulla.
- Inoltre fornisce alcuni servizi:
  - Caricamentoe controllo con un class loader e un security manager
  - Gestione di oggetti remoti, replicati e persistenti
  - Attivazione automatica degli oggetti e multithreading
  - Garbage collection di oggetti remoti con un meccanismo di conteggio delle reference

6.9.1 Trasferimento dei parametri

Qualsiasi tipo primitivo è passato per valore, insieme agli oggetti non remoti SOLO SE SERIALIZZABILI

Esiste anche trasferimento per reference? Sì, i riferimenti ad oggetto remoto sono passati per valore per permettere invocazioni remote. In particolare:

1. La classe **java.rmi.server.UnicastRemoteObject** è una reference class fatta apposta per lo scopo

2. Implementa le interfacce Remote e Serializable

Sì ok, però si parla da entrambe le parti di passaggio per valore.. Perchè? Tutto merito della serializzazione

6.9.2 Concetto di serializzazione

E' molto semplicemente la conversione di un oggetto, un dato, un valore, in una sequenza di 0 ed 1, un flusso, da un oggetto ad un flusso. Il processo inverso è chiamato **deserializzazione**, o anche noto come **casting**. Ricevi un flusso, gli dai una forma, fine.

A che serve? In rete qualsiasi cosa è passata serializzata, ma occorre che si definiscano degli oggetti persistenti.

Il meccanismo di loafding dinamico di Java permette di passare solo le info essenziali sullo stato, infatti la **descrizione della classe** può essere caricata a parte.

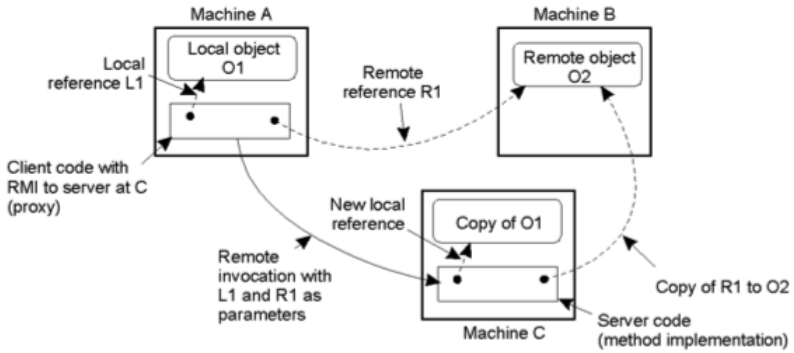
I tipi base sono serializzabili in modo nativo, mentre per classi più complesse serve implementare l'interfaccia **Serializable**, che ti fa ridefinire

1. private void writeObject(java.io.ObjectOutputStream out) throws IOException;

2. private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;

6.9.3 Identificazione degli oggetti

Come può una macchina A ricevere la reference remota dell'oggetto remoto O2?



1. Prima di tutto usando nomi assegnati dall'utente e un **directory service** per convertirli in **reference operative**

2. I **directory service** oppure anche **naming service** devono essere disponibili ad un host e porta noti (well-known addresses)

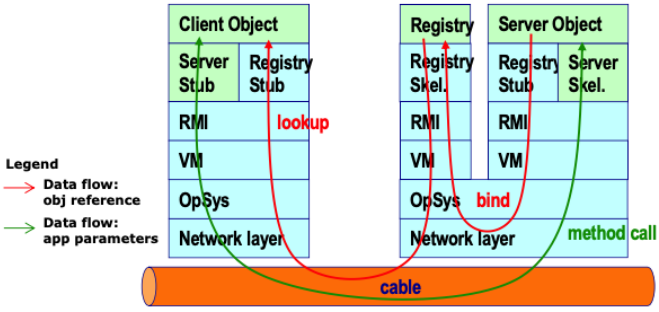
3. Infatti a questo scopo RMI definisce un **rmuregistry** che sta su ogni macchina che ospita oggetti remoti (per convenzione sulla porta 1099)

4. RMI attiva un servizio di ascolto.. per quel servizio (Sorry per la ripetizione, non so in che altro modo possa dirlo)

Esiste quindi una classe Naming? Sì esatto, e dà diretto accesso alle funzionalità dei RMI Registry

6.9.4 Architettura in dettaglio

- Il server pubblica il reference e il nome dell'oggetto remoto nel Registry invocando il metodo bind
- Il client ottiene il reference all'oggetto invocando il metodo lookup
- Il client accede all'oggetto remoto il reference e i nomi dei metodi remoti



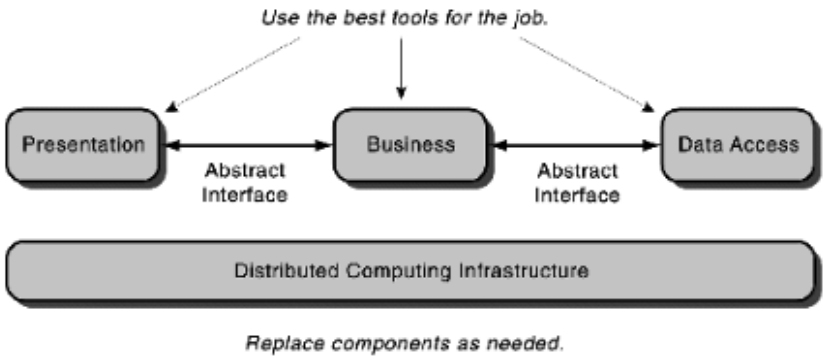
Capitolo 7

Ajax - Javascript

7.1 Architettura Multitier

Riprendendo cos'è l'architettura multi-layer, qui fondamentalmente si parla di applicazioni composte da più componenti che collaborano per eseguire un compito.

Prendiamo una tipica scomposizione in 3 livelli:



In un'applicazione scomposta su questi 3 livelli, la parte business agisce come server per la presentation, mentre agisce come client per la componente data

7.2 RIA - Rich Internet Application

Quali sono le caratteristiche del web?

1. Le pagine web si **ricaricano** e non **aggiornano**
2. Gli utenti aspettano che si carichi l'intera pagina anche se serve solo un dato
3. Si hanno restrizioni su singole request/response

Ajax, da titolo, che cos'è? E' una tecnica di sviluppo per creare delle applicazioni web interattive (in senso di pattern programmativo). **Ajax** sta per: **Asynchronous Javascript and XML**, quindi

- Asynchronous: rende asincrona la comunicazione tra browser e web server
- Javascript: E' il cuore del codice tramite cui funzionano le applicazioni Ajax, ed è di supporto per la comunicazione con applicazioni server
- XML: è solo usato per trasferire i dati per costruire pagine web identificando i campi per il successivo uso nel resto dell'applicazione (Json è il più comune)
- DOM: Utilizzato tramite JavaScript per manipolare sia la struttura della pagina HTML che le risposte XML del server

Quali sono le componenti?

-

C'entra qualcosa il web 2.0? Sì, se no non lo menzionavo. Il web 2.0 fa sì che il web diventi una piattaforma, in cui l'esperienza utente è ricca, ed il contenuto viene direttamente dall'utente stesso (con la nascita dei forum)

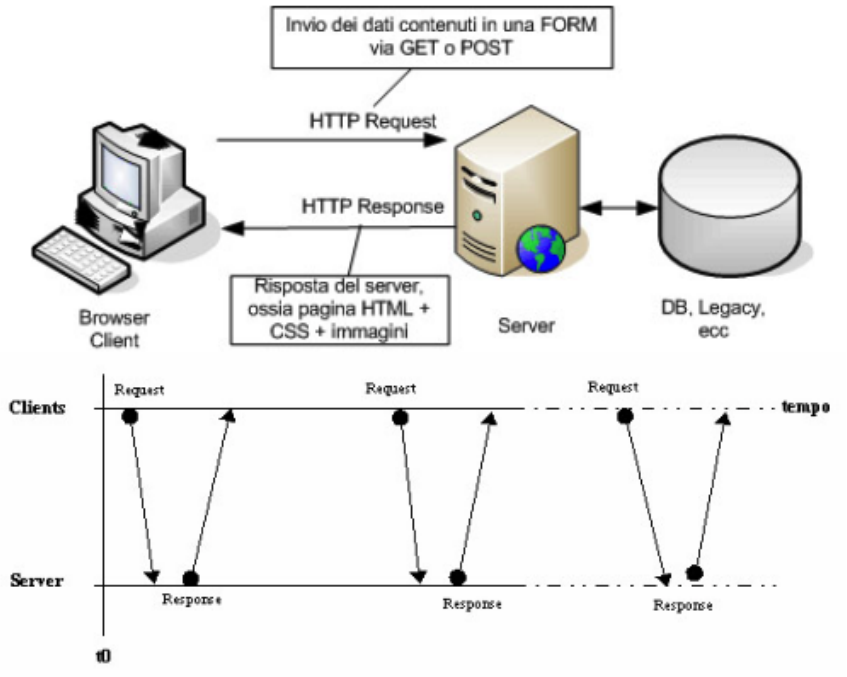
Okay, qualche esempio di uso dell'Ajax?

1. Form validati in corso di compilazione
2. Autocompletamento dei campi dei form
3. Controlli UI sofisticati

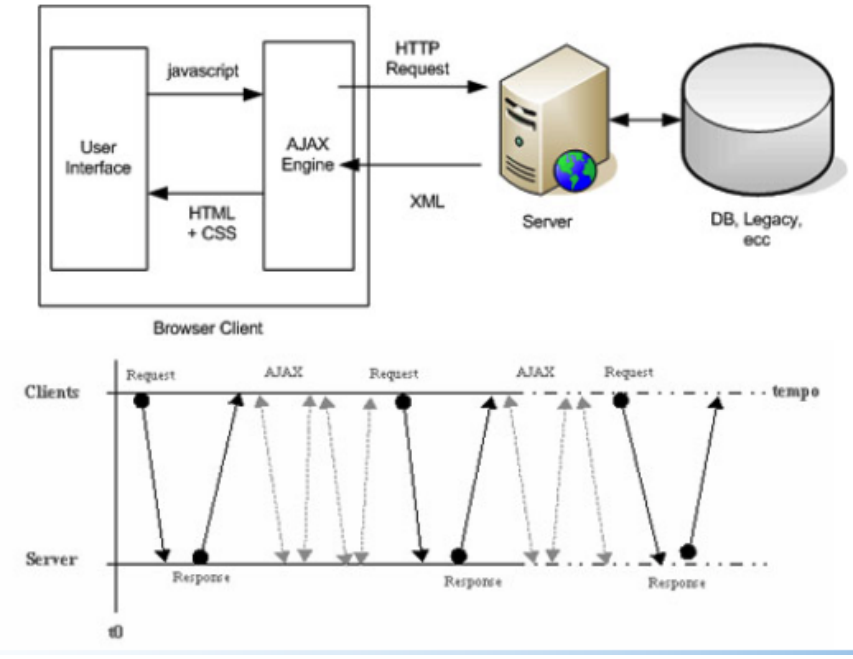
Componenti Ajax

- HTML
- DOM
- XML e XSLT
- Oggetto XMLHttpRequest
- Javascript

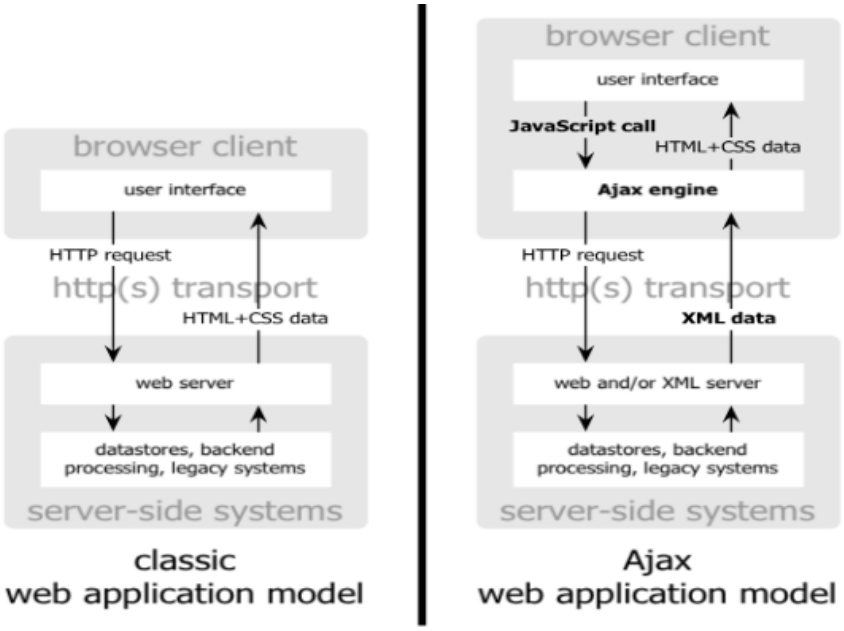
7.2.1 Architettura web classica



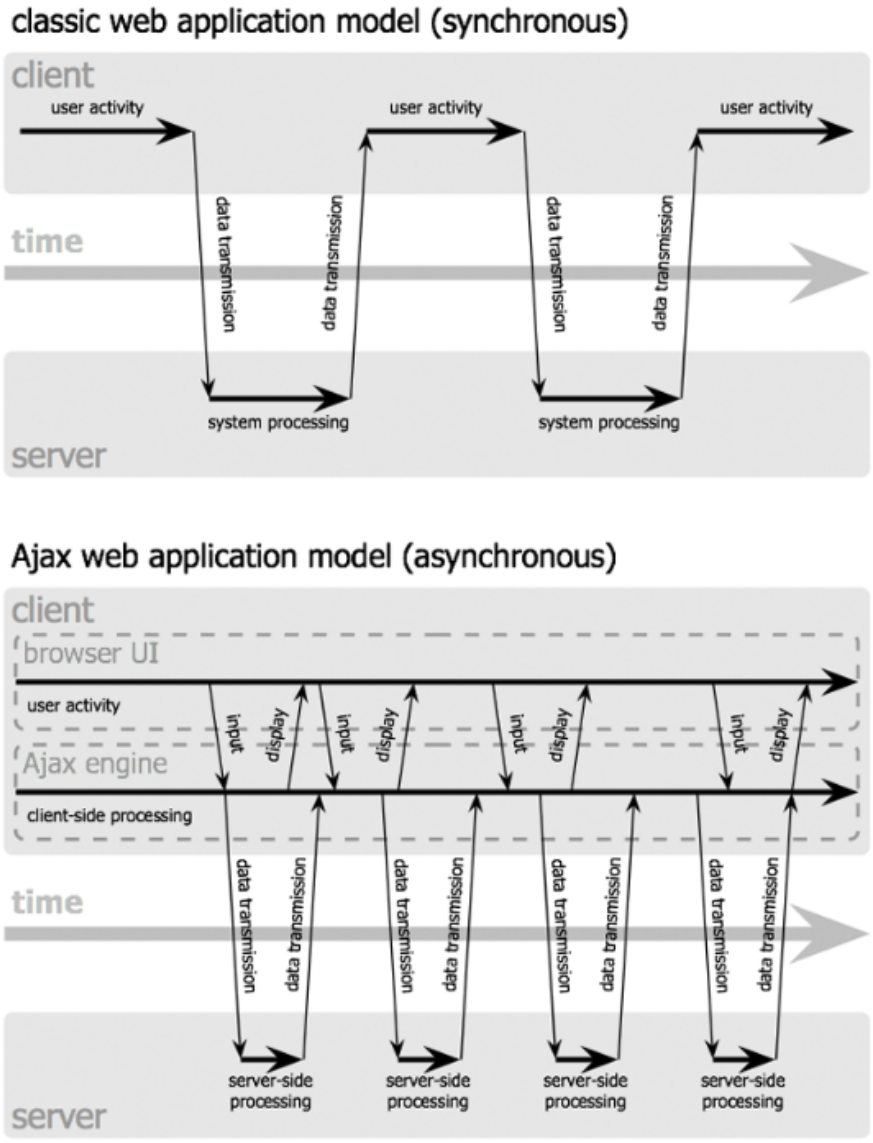
7.2.2 Architettura web dinamica



7.2.3 Applicazioni web e Ajax



7.2.4 Processo request



7.2.5 Principali problemi

1. Rottura del tasto back dei browser, poichè ci sono degli IFrames che possono invocare cambiamenti in grado di modificare la cronologia
2. Cambiamenti di alcune parti della pagina inaspettate
3. L'aggiunta ai segnalibri di un determinato stato risulta difficoltosa, perchè JS genera la pagina, NON il server
4. L'incremento della dimensione, o lunghezza del codice ha un brutto impatto sui tempi di caricamento del browser, ed i tempi di risposta si dilatano
5. Difficile da debuggare
6. La sorgente è visualizzabile, quindi il plagio è estremamente facile
7. Il server si fa carico del lavoro, poichè una richiesta anonima è pesante da gestire

7.2.6 Trasmissione dei dati

La trasmissione dei dati tra server e applicazioni RIA è un aspetto critico, infatti esistono diverse alternative:

1. SOAP (Simple Object Access Protocol)
2. XML-RPC (Remote Precedure Call con XML per l'encoding)
3. JSON (Javascript Object Notation)
4. AMF (Action Message Format, basasto su SOAP)

La scelta del formato avrà influenza sulle performance e sulla struttura dell'applicativo inoltre un formato richiede del tempo per

1. Predisporre i dati lato server
2. Trasferire i dati
3. Effettuare il parsing dei dati
4. Effettuare il rendering dell'interfaccia stessa

7.3 Javascript

Forse è già stato detto, ma va ripetuto cos'è un linguaggio di scripting, ossia un linguaggio per l'automazione di compiti eseguibili da un utente umano all'interno di un ambiente software. Per ogni dominio di applicazione c'è un linguaggio praticamente, anche qui esistono i **general purpose** come Python, e per l'appunto Javascript.

Caratteristiche tipiche

1. Semplicità
2. Specificità
3. Intepretazione (Non è codice compilato, ma sempre interpretato)

LISP (nella variante AutoLISP) è attualmente incluso in AutoCAD, quindi AutoCAD è un simulatore di parentesi, mica male! Ed in tutto questo, Javascript? E' un linguaggio pensato per eseguire script in un browser web **lato client** , e soprattutto per l'interazione con l'utente, validazione dei dati nei form etc.

JS è dinamico, fortemente tipizzato e con una sintassi simile a Java e C, che tra l'altro questa affermazione è ricorsiva, perchè Java stesso è simile a C. Per vedere come funziona Javascript c'è W3Schools, è il meglio. Quello che ci interessa di nuovo è il jQuery.

Per giochicchiare con Javascript andate su JSFiddle

7.4 jQuery

E' un framework JavaScript che rende più semplice la scrittura di applicazioni web offrendo diverse funzionalità.

1. Manipolazione HTML/DOM e CSS
2. Metodi per eventi HTML
3. Effetti e animazioni
4. Supporto per AJAX
5. Plugins vari

Essendo il più utilizzato manutenuto e testato si è deciso di usare questo, soprattutto perchè è Open, compatto ed estendibile.

7.4.1 Elementi base di jQuery

La sintassi è specificamente orientata a permettere una rapida selezione di elementi del documento HTML, di base un comando si fa con: $(selector).action()

In cui il $ definisce l'accesso a funzioni jQuery

Selector è usato per specificare una query per selezionare una parte del documento HTML mentre action.. Beh è ciò che viene eseguito sull'elemento stesso.

Menzione speciale per la porzione:

$(document).ready(function()) [...]); // end ready

Praticamente indica che vogliamo effettuare l'azione **ready** sull'oggetto **document** passando come parametro una funzione anonima, specificata per esteso

L'azione ready invece specifica che il parametro ricevuto, una funzione, va ad essere richiamata quando il documento è 'pronto', ovvero caricato completamente.

Altri elementi utili:

- Ciò che vogliamo fare è selezionare le porzioni di documento presenti ad esempio negli span di classe pq, clonarli dando allo steso tempo lo stile desiderato

  Il codice risulterà così: $('span.pq').each(function() [...]) //end each

- Praticamente specifichi una funzione anonima che si applica ad ogni span di classe pq, tipo quelle che si scrivevano in LISP, le lambda

Capitolo 8

Concorrenza in Java

Cosa significa che un software è concorrente? Molto semplicemente che ha diversi controlli di flusso in contemporanea, cioè lo stesso software ha più esecuzioni in contemporanea. Alcune funzioni ricorsive se effettuate con i Thread possono risolvere il problema molto prima.. O riempirvi subito tutta la memoria.

8.1 Perchè software concorrenti?

Perchè si sfruttano i sistemi multi-core, e perchè si divide il carico di lavoro, facendo in modo che si esegua più velocemente. Una task che con un core ci impiega 10, con 2 core ne impiega 5, e così via. **Ma non solo.**

Un altro motivo è il fatto che in questo modo se si dovesse bloccare uno dei thread, non crasha tutto l'intero software (questo dimostra che la UI è un thread scollegato da ciò che sta sotto)

Ultimo ma non ultimo: Per strutturare il programma in modo più adeguato, in particolare programmi che hanno diverse cose da tenere sotto controllo, che gestiscano diversi tipi di eventi (Tipo Overwatch, per ogni giocatore ci sarà sotto un Thread), per ogni countdown degli oggetti etc.

Come vengono visti dal sistema operativo? I processi non condividono risorse tra loro, a parte l'esecuzione dalla stessa CPU, ma a questo punto per comunicare devono usare file, syscall, servizi, o middleware etc., inoltre i thread di uno stesso processo condividono stesso spazio di indirizzamento e oggetti in memoria. Non mi sto a dilungare su come si facciano i Thread in java, vi basti sapere che ci son due modi.

1. Implementazione dell'interfaccia Runnable
2. Estensione della classe Thread

Sono praticamente la stessa cosa, son due metodi equivalenti.

8.2 Problemi della concorrenza

Avere più processi che operano con le stesse risorse potrebbe rivelarsi un problema, e non indifferente. In primo luogo c'è il problema delle risorse condivise.

8.2.1 Problema di accesso a risorse condivise

Ipotizziamo di avere due processi che operano sulla stessa variabile, potrebbero avvenire modifiche sbagliate, o uno dei processi potrebbe riscontrare un'incongruenza, quindi si genera un problema di consistenza di quella variabile.

A cosa è dovuto il problema? Quando le operazioni sulla variabile non sono atomiche, diventano interrompibili anche a metà della loro esecuzione, questo causa il problema di prima. Come si risolve? **Mutua Esclusione**

8.2.2 Mutua esclusione

Il concetto è questo: per mutua esclusiva, se utilizzo una risorsa, io processo sarò l'unico ad utilizzarla fino a fine esecuzione. Come lo posso fare? con la keyword **synchronized** che letteralmente si occupa di fare questo, poichè Java associa un intrinsic lock ad ogni oggetto che abbia almeno un metodo synchronized.

Direttamente da Repl.it c'è Questo esempio proposto direttamente dal Vizzari. (Se non dovesse esserci, ci sarà un file.java dentro la cartella di SD)