Prof. Paolo Bellavista A.A. 2023/24

Tecnologie Web T

Valentina Bacchelli

INTRODUZIONE

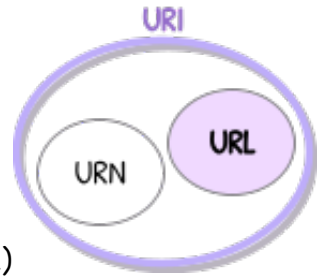
- WWW nasce nel 1989, è una applicazione **on top** dei protocolli internet
  - prima condivisione documenti basata su **FTP**
  - obiettivo = condividere documenti statici **ipertestuali** (con link monodirezionali) basandosi su protocolli semplici
- **cliente** = browser, **servitore** = http server, client sono attivi e server passivi
- Ancora oggi in uso HTTP/1.1 (basato su TCP), mentre HTTP/2 e HTTP/3 (basato su UDP) sono add-on relativamente limitati
- **Ipertesto** crea rete (o grafo) in cui i documenti sono nodi
  - lettura può essere non lineare
  - si parla di **ipermedia** quando ci sono elementi multimediali (immagini, video,...)
  - ipertesti nati molto prima del web

- idea base: SEMPLICITÀ
  - 3 elementi concettuali: - localizzare il documento (**url**)
    - protocollo per accedere alle risorse (**http** = hypertext transfer protocol)
    - linguaggio per descrivere documenti ipertestuali / costruire pagine (**html**)
  - 2 elementi fisici: - **server** per erogare risorse che costituiscono pagine
    - **client** per visualizzare documenti e consentire navigazione

WWW = URL + HTTP + HTML

URI e URL

- Necessità di identificare il server in grado di fornirci una pagina, di individuare la risorsa che vogliamo e accedervi
- URI = Uniform Resource Identifier
  - stringhe con sintassi definita (che dipende da schema): <scheme>:<scheme-specific-part>
  - esiste sottoinsieme di URI per rappresentare relazioni gerarchiche in uno spazio di nomi:
    <scheme>://<authority><path>?<query> (certe parti possono essere omesse)



  - divisi in URN e URL
    - URN = Uniform Resource Name contiene nome univoco per la risorsa ma non la sua locazione
      → serve altro servizio per identificare DOVE si trova risorsa
      → esempio: ISBN dei libri è un URN
    - URL = Uniform Resource Locator contiene locazione della risorsa, specifica protocollo per il trasferimento (questo implica che non viene usato necessariamente solo HTTP)

→ il resto del nome dipende dal protocollo usato

- In generale per protocolli HTTP, HTTPS, FTP e altri simili la struttura generale è:

  <protocol>://[<username>:<password>@]<host>[:<port>][/<path>[?<query>][#fragment]]

  ○ protocol indica il protocollo usato

  ○ attenzione: username e password inseriti così passano in chiaro sul web (metodo mai usato)

  ○ le parentesi quadre [] indicano che quella parte può essere omessa

  ○ host indica l'ip della macchina server → come faccio se ho più macchine server? diverso ip, cambierebbe url

  ○ path è il path fisico nel file system del server

  ○ fragment indica parte specifica all'interno della pagina

- questo non funziona, ad esempio, per posta elettronica
- URI opaca = non soggetta ad altre operazioni di parsing (es. mailto:...)
- URI gerarchica = serve parsing per portarla nella sua forma finale
  - ○ es. se manca path si vede in automatico la homepage con path default
  - ○ se c'è path ma non host può essere sottinteso "sulla stessa macchina" (codebase = risoluzione)
  - ○ relativizzazione processo inverso alla risoluzione
  - ○ simboli speciali come .. o ~ (normalizzazione)

HTTP

- modello di interazione SEMPLICE
- ricorda: web pensato come servizio semplice CLIENTE/SERVITORE
- HTTP = Hypertext transfer protocol, protocollo di liv applicativo utilizzato per trasferire risorse web (pagine o elementi di pagine) da server a client
  - ○ Protocollo applicativo → LIVELLO 7 iso/osi

- gestisce richieste da client a server (URL) e risposte restituite al client (pagine) → protocollo REQUEST - RESPONSE

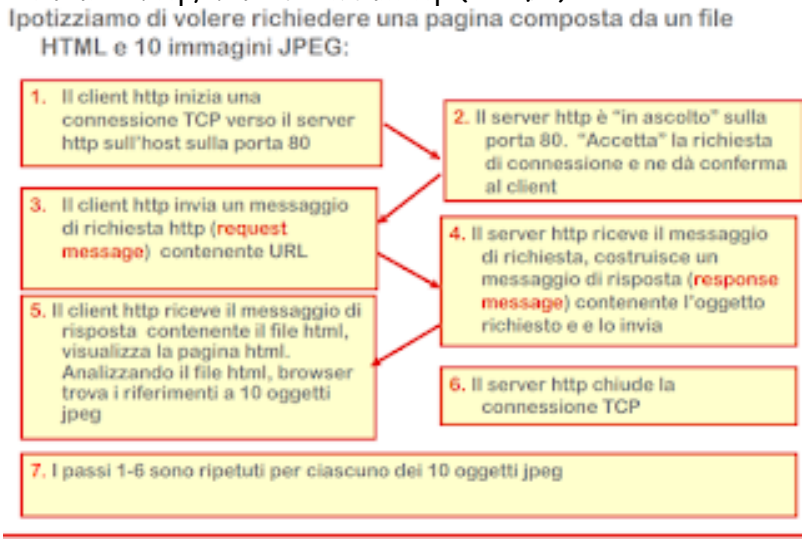
- Protocollo STATELESS = nè server nè client mantengono info su messaggi passati
- esempio tcp: handshake è stato usato per gestire connessione
- CLIENT: programma applicativo che stabilisce connessione per inviare richieste
- SERVER: programma applicativo che accetta connessioni per ricevere richieste e restituire risposte

- CONNESSIONE: circuito virtuale stabilito a liv di trasporto tra due app per comunicazione
- MESSAGGIO: unità base di comunicazione http, specifica sequenza di byte concettualmente atomica
    - request: richiesta
    - response: risposta
- resource: oggetto di tipo dato univocamente definito
- URI: identificatore univoco per una risorsa
- Entity: rappresentazione di una risorsa

http v1.0

- protocollo request-response, stateless, one shot

- basato su tcp, usa solo stream tcp (no UDP)



- Segue uno schema di questo tipo:

    - server rimane in ascolto (server passivo), tipicamente sulla porta 80

    - client apre una connessione TCP sulla porta 80 (cliente attivo, da quale porta locale?)

    - server accetta la connessione (possibili più connessioni in contemporanea?)

    - client manda una richiesta

    - server invia la risposta e chiude la connessione

- quindi ogni connessione si chiude dopo 1 richiesta e 1 risposta, non si fanno più scambi → connessione NON PERSISTENTE

http v1.1

- stessa connessione http utilizzabile per più scambi

- possibile specificare coppie multiple di richiesta e risposta nella stessa connessione

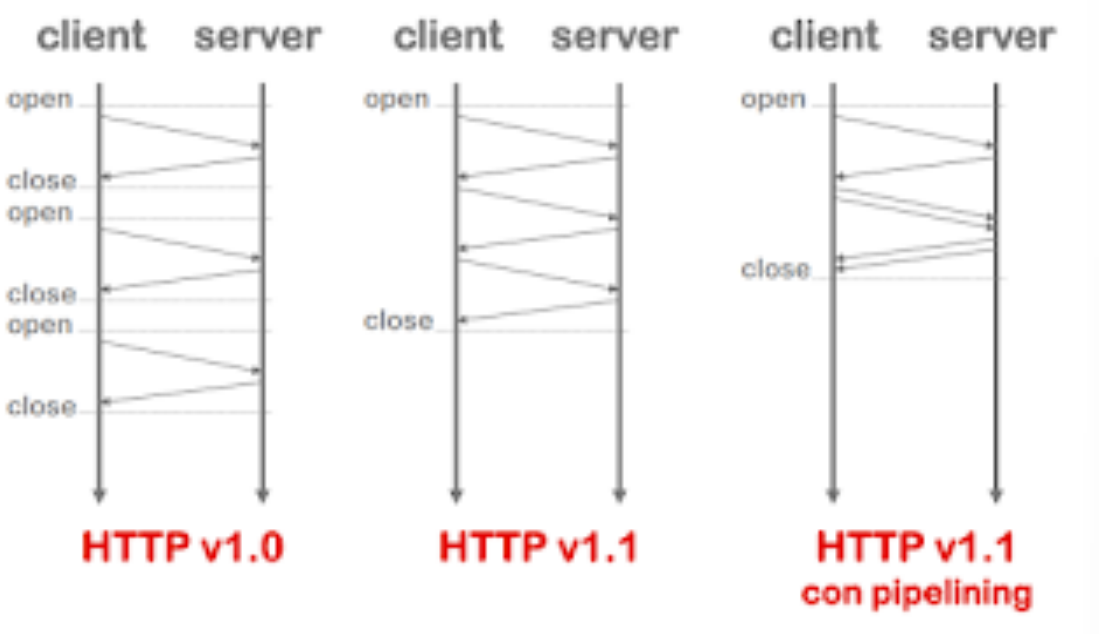
- connessione PERSISTENTE → connessione tcp lasciata aperta dal server dopo l'invio della risposta, può ricevere altre richieste su di essa

- connessione chiusa dal server quando specificato in header messaggio (richiesta dal cliente) o non viene usata da un certo tempo (time out)

pipelining

  - possibile mandare più richieste senza che sia terminata ricezione risposte precedenti

  - risposte però devono essere in stesso ordine delle richieste perché non c'è modo specifico di associare richiesta e risposta



- Obiettivo fondamentale HTTP/2: miglioramento performance complessiva con full backward compatibility con HTTP 1.1

- request-response multiplexing
- header compression
- server push

- HTTP/2 è basato su SPDY, protocollo di cosiddetto open networking promosso da Google

- Obiettivo fondamentale di HTTP/3: Miglioramento di HTTP tramite integrazione con protocollo di trasporto QUIC (invece del classico TCP)
  - stream multiplexing
  - controllo di flusso per-stream
  - realizzazione di connessioni con bassissima latenza

Sostanzialmente mapping della semantica HTTP su QUIC

- HTTPS è HTTP ma con connessione sicura

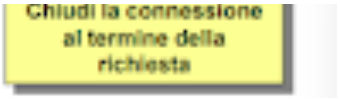
- Un messaggio HTTP è definito da due strutture:
  - Message Header: contiene tutte le informazioni necessarie per identificazione del messaggio (più in generale tutte le intestazioni del messaggio)
  - Message Body: contiene i dati trasportati dal messaggio

- Esistono schemi precisi (standard, definiti e non modificabili) per ogni tipo di messaggio relativamente a header e body

- I messaggi di Response contengono i dati relativi alle risorse richieste (tipicamente una pagina

html)

- I dati sono codificati secondo il formato specificato nell'header
- Solitamente sono in formato MIME (Multipurpose Internet Mail Extensions)
- header formati da insiemi di coppie (nome:valore) che specificano caratteristiche del mess trasmesso o ricevuto
- header generali della trasmissione
    - data, codifica, versione, tipo di comunicazione,...
- header relativi all'entità trasmessa
    - content-type, content length, data scadenza,...
- header riguardo richiesta
    - chi fa richiesta, a chi, che caratteristiche accetta, autorizzazione,...
- header della risposta
    - che server dà risposta, autorizzazione necessaria,...
- messaggi sono in formato ascii → leggibili



metodi di richiesta

GET

- Richiesta di risorsa al server
    - frequente: usato quando si fa click su link ipertestuale o ricerca di url in browser
- si passano parametri (parte <query>)
- max lunghezza url limitata

POST

- richiedere risorsa
- dettagli per identificare non sono in url ma nel body del mess
- no limiti lunghezza
- usato per sottomettere dati di una form html ad un'applicazione cgi sul server
- trasmissione senza creazione di risorsa sul server

PUT

- chiede di memorizzare risorsa sul server a url specificato
- trasmissione info da client a server
- si crea risorsa o si sostituisce se esiste già
- argomento di put è la risorsa che verrà restituita quando in seguito si farà get allo stesso url

DELETE

- richiede cancellazione risorsa a url specificato
- di solito disabilitato su server pubblici

HEAD

- simile al get ma no body, solo header relativi
- verifica url
    - validità: risorsa esiste e lunghezza ≠ 0
    - accessibilità: non è richiesta autenticazione

OPTIONS

- richiedere info sulle opzioni disponibili per comunicazione

TRACE

- invocare loop back remoto a livello app del messaggio di richiesta
- consente a client di vedere cosa ha ricevuto il server (per testing)

messaggi di risposta

- HTTP 1.0: server chiude la connessione al termine della richiesta

- HTTP 1.1: server mantiene aperta la connessione oppure la chiude se si mette la clausola: `Connection: close`

codici di stato

- status code = numero di 3 cifre, prima è classe di risposta, altre due indicano risposta specifica

- 5 classi:

  - 1xx informational: risposta temporanea durante svolgimento richiesta (sconsigliata a partire da http 1.0)

  - 2xx successful: richiesta ricevuta, capita e accettata da server

  - 3xx redirection: richiesta ricevuta e capita ma richieste altre azioni dal client prima di portarla a termine

  - 4xx client error: richiesta non può essere soddisfatta perché sbagliata (mancanza di autorizzazioni o errore di sintassi)

  - 5xx server error: richiesta magari corretta ma errore lato server, interno (del server o di app cgi)

Esempi di codici di stato

- `100 Continue` (se il client non ha ancora mandato il body, deprecated da HTTPv1.0)
- `200 Ok` (GET con successo)
- `201 Created` (PUT con successo)
- `301 Moved permanently` (URL non valida, il server conosce la nuova posizione)
- `400 Bad request` (errore sintattico nella richiesta)
- `401 Unauthorized` (manca l'autorizzazione)
- `403 Forbidden` (richiesta non autorizzabile)
- `404 Not found` (URL errato)
- `500 Internal server error` (tipicamente un CGI mal fatto)
- `501 Not implemented` (metodo non conosciuto dal server)

- (reti) server sequenziale vs parallelo:
    - Server sequenziale deve aspettare le richieste che sono in coda e le serve una alla volta (ciclo infinito, prende una richiesta, la serve, e dà risposta passa alla richiesta successiva ...
    - Server concorrente deve essere capace di servire più di una richiesta alla volta, e quindi estrae una richiesta in coda, ma prima di terminare il servizio e dare risposta, può (e deve) estrarre anche altre richieste in coda e servirle contemporaneamente → prende una richiesta comincia a servirla e prima di dare risposta può lavorare anche su richieste precedenti o attivandone di successive...
    - Il ciclo di lavoro sequenziale è molto semplificato, mentre il ciclo di lavoro concorrente (anche detto parallelo) è più complesso
    - Un server concorrente può essere un unico processo con più attività o parallelo (molte attività e processi distinti, pesanti o leggeri)
- Domande:
    - Può essere sufficiente avere un pool di processi/thread su un unico server ad altissime prestazioni?
    - Come andrebbe gestita opportunamente una soluzione con n server disponibili in parallelo?
    - Quali problemi?

cookie

- COOKIE: struttura dati che si muove come un token dal client al server e viceversa
    - generati sia da client che da server
    - una volta creati vengono passati ad ogni trasmissione
    - forniscono supporto per MANTENIMENTO DI STATO visto che http è stateless
    - coppia nome:valore
    - implementati come header dei messaggi di request o response

- implementati come header dei messaggi di request o response

- collezione di stringhe

  - key= identificatore univoco nel dominio:path

  - value= valore associato al cookie (stringa max 255 caratteri)

  - path: posizione nell'albero di un sito a cui è associato (di default /)

  - domain= dominio in cui è stato generato

  - max-age= (optional) numero di secondi di vita per far SCADERE SESSIONE

  - secure (opzionale)= trasferito se e solo se il protocollo è sicuro (https). non molto usato

  - version= identifica versione protocollo gestione dei cookie

autenticazione

restringere accesso a risorse solo ad utenti autorizzati: 4 metodi

- filtro su set di ip (mai usato, indirizzo deve essere pubblico e statico, possibile spoofing → fingere ip fasullo)

- form per richiedere user e pw → usa metodo post , in chiaro

- http basic → status code 401 authorization required richiede user e pw, nuova request del client invia user e pw, server risponde nuovamente stavolta soddisfando richiesta. testo passato in chiaro! dentro al messaggio

- http digest → caduta in disuso recentemente , invio di pw dopo hash

sicurezza (https)

- proprietà:
    - confidenzialità: se attaccante trova mess non lo capisce, è criptato
    - integrità: se attaccante modifica un nostro mess è comprensibile che è stato modificato e non è integro, viene ignorato
    - autenticità: se mess mandato ha info sul mittente, essa è vera. (attaccante non si può fingere qualcun altro come mittente)
    - non ripudio
- prime 3: ssl / tls - SICUREZZA DEL CANALE DI TRASPORTO
    - ssl: secure sockets layer
    - tls: transport layer security → sostituisce ssl, è alla base di http
    - livello che si occupa di gestione di confidenzialità, autenticità ed integrità comunicazione tra http e tcp
    - basato su crittografia a chiave pubblica: public key + private key

- **Architetture più distribuite e articolate per il Web:**
    - Proxy: Programma applicativo in grado di agire sia come Client che come Server al fine di effettuare richieste per conto di altri Clienti. Le Request vengono processate internamente oppure vengono ridirezionate al Server. Un proxy deve interpretare e, se necessario, riscrivere le Request prima di inoltrarle

○ Gateway: Server che agisce da intermediario per altri Server. Al contrario dei proxy, il gateway riceve le request come se fosse il server originale e i Client non sono in grado di identificare che Response proviene da un gateway. Detto anche reverse proxy o server-side proxy

○ Tunnel: Programma applicativo che agisce come "blind relay" tra due connessioni. Una volta attivo (in gergo "salito") non partecipa alla comunicazione HTTP

→ ad esempio per caching distribuito:

- Idea di base: memorizzare copie temporanee di documenti Web (es. pagine HTML, immagini) al fine di ridurre l'uso della banda ed il carico sul server

- Una Web cache memorizza i documenti che la attraversano. L'obiettivo è usare i documenti in cache per le successive richieste qualora alcune condizioni siano verificate

- Tipi di Web cache:

  ○ User Agent Cache: mantiene una cache delle pagine visitate dall'utente

    ■ era molto importante in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga

    ■ comunque ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente ma anche per ridurre latenze dovute al caricamento di elementi statici (icone, sfondi, ...)

  ○ Proxy Cache

    ■ Forward Proxy Cache: servono per ridurre le necessità di banda (es. rete locale aziendale, Università, ecc.), proxy intercetta il traffico e mette in cache le pagine. Successive richieste non provocano lo scaricamento di ulteriori copie delle pagine al server

    ■ Reverse (o server-side) Proxy Cache: gateway cache, operano per conto del server e consentono di ridurre il carico computazionale delle macchine. I client non sono in grado di capire se le pagine arrivano dal server o dal gateway. Internet

Caching Protocol per il coordinamento tra diverse cache. Base per content delivery network (CDN)

- HTTP definisce meccanismi che possono avere effetti collaterali positivi per la gestione "lazy" dell'aggiornamento cache

  - Freshness: controllata lato server da Expires response header e lato cliente da direttiva Cache-Control: max-age

  - Validation: può essere usato per controllare se un elemento in cache è ancora corretto, ad es. nel caso in cui sia in cache da molto tempo (ad es. tramite richieste HEAD)

  - Invalidation: è normalmente un effetto collaterale di altre request che hanno attraversato la cache. Se per esempio viene mandata una POST, una PUT o una DELETE a un URL il contenuto della cache deve essere e viene automaticamente invalidato

HTML

- Hypertext Markup Language

- È il linguaggio utilizzato per descrivere le pagine che costituiscono i nodi dell'ipertesto

- È un linguaggio di codifica del testo del tipo a marcatori (markup) = formalismo con cui rappresentare un documento su supporto digitale in modo che sia trattabile dall'elaboratore in quanto testo

- formalismi più elementari = sistemi di codifica dei caratteri

- formalismi più elementari → sistemi di codifica dei caratteri

    - corrispondenza biunivoca tra elementi di una collezione ordinata di caratteri e un insieme di codici numerici

    - Si ottiene così un coded character set che di solito si rappresenta in forma di tabella (code page o code table)

    - Per ciascun coded character set si definisce una codifica dei caratteri (character encoding)

    - La codifica mappa una o più sequenze di byte (8 bit) a un numero intero che rappresenta un carattere in un determinato coded character set

    - numero di caratteri rappresentabili in un certo coded character set è determinato dal numero di bit utilizzati per codificare ogni singolo carattere

- per un testo non basta codifica caratteri: un testo è un oggetto complesso caratterizzato da molteplici livelli strutturali che non si limitano alla sequenza di simboli del sistema di scrittura

    - Si parla propriamente di linguaggio di codifica testuale solo in riferimento ai linguaggi che consentono la rappresentazione o il controllo di uno o più livelli strutturali di un documento testuale

    - Tali linguaggi vengono correntemente denominati linguaggi a marcatori (mark-up language)

- Un linguaggio di mark-up è composto da:

    - un insieme di istruzioni dette tag o mark-up (marcatori) che rappresentano le caratteristiche del documento testuale

    - una grammatica che regola l'uso del mark-up

    - una semantica che definisce il dominio di applicazione e la funzione del mark-up

- I marcatori vengono inseriti direttamente all'interno del testo cui viene applicato

- Ogni tag è a sua volta costituito da sequenza di caratteri, preceduta da caratteri speciali che la delimitano e che permettono di distinguere facilmente il testo dai marcatori

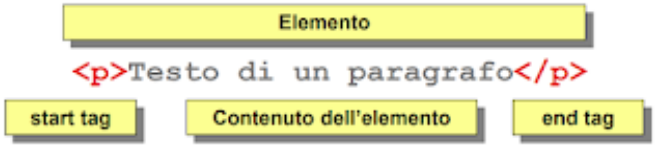
- Tradizionalmente i linguaggi di mark-up sono stati divisi in due tipologie:
    - linguaggi procedurali/imperativi: mark-up specifica quali operazioni un dato programma deve compiere su un documento elettronico per ottenere una determinata presentazione (Tex, LateX)
    - linguaggi dichiarativi/descrittivi: mark-up descrive struttura del documento testuale identificandone i componenti (SGML, HTML, XML) → struttura editoriale, componenti (CONTENT OBJECT) organizzati gerarchicamente

SGML

- Standard Generalized Markup Language
- meccanismo flessibile e portabile per rappresentare documenti elettronici proposto da Charles Goldfarb
- Un documento SGML comprende oggetti di varie classi chiamati elementi (capitoli, titoli, riferimenti, oggetti grafici, etc.)
- SGML identifica gli estremi degli elementi tramite tag iniziali e tag finali
- Non contiene sequenze di istruzioni di formattazione
- Gli elementi sono organizzati in una gerarchia: un capitolo contiene un titolo ed una o più sezioni che a loro volta contengono altri elementi, ecc.
- HTML = applicazione/semplificazione di SGML, ovvero un linguaggio per la rappresentazione di un tipo di documento SGML
- Ideato da Tim Berners-Lee a inizio '90
- Tramite HTML è possibile realizzare documenti con una struttura semplice contenenti: testo, immagini e contenuti multimediali, oggetti interattivi e connessioni ipertestuali ad altri documenti
- Oltre a descrivere il contenuto, HTML associa anche significati grafici agli elementi che definisce! → istruzioni più o meno precise su come rendere graficamente gli elementi che definisce
- unione di elementi di formattazione però crea problemi
- HTML 4.01 risulta ancora il linguaggio con cui sono scritte la maggioranza delle pagine Web attuali (assieme a XHTML di cui parleremo in seguito), molto di ciò che il linguaggio HTML 5 offre rispetto a HTML 4.01 è opzionale
- HTML 4.01 prevede tre varianti:
    - Strict, in cui gli elementi deprecati sono vietati
    - Transitional, in cui gli elementi deprecati sono ammessi
    - Frameset, in cui sono ammessi anche i frame e gli elementi collegati
- Nel maggio 2000 l'HTML 4.01 Strict è diventato standard ISO/IEC con il codice 15445:2000

Tag

- I tag HTML sono usati per definire il mark-up di elementi HTML



- Sono preceduti e seguiti rispettivamente da due caratteri "<" e ">" (parentesi angolari)
- Sono normalmente accoppiati; un esempio è dato da: e , detti rispettivamente start tag ed end tag
- Il testo tra start tag ed end tag è detto contenuto dell'elemento
- Un documento HTML contiene quindi elementi composti da testo semplice delimitato da tag

- HTML rispetta in maniera poco rigorosa le specifiche SGML
- Ammette elementi senza chiusura come
- I tag non sono case sensitive
- L'apertura e chiusura di tag annidati può essere "incrociata":

  Testo corsivo grassetto
- Esistono però delle buone pratiche che è bene rispettare e che diventano un obbligo in una versione più rigorosa del linguaggio chiamata XHTML:
  - Chiudere sempre anche i tag singoli
  - Tag in minuscolo
  - Apertura e chiusura senza incroci (in teoria non ammessi ma tollerati)

Entity

- HTML definisce un certo numero di entità (entity) per rappresentare i caratteri speciali senza incorrere in problemi di codifica:
  - Caratteri riservati a HTML (<, >, &, ", ecc.)
  - Caratteri non presenti nell'ASCII a 7 bit: (& → &, " → ")

Attributi

- Un elemento può essere dettagliato mediante attributi
- Gli attributi sono coppie "nome = valore" contenute nello start tag con una sintassi di questo tipo:

  <tag attrib1='valore1' attrib2='valore2'>
- I valori sono racchiusi da apici singoli o doppi
- ! Gli apici possono essere omessi se il valore non contiene spazi

Tipi MIME

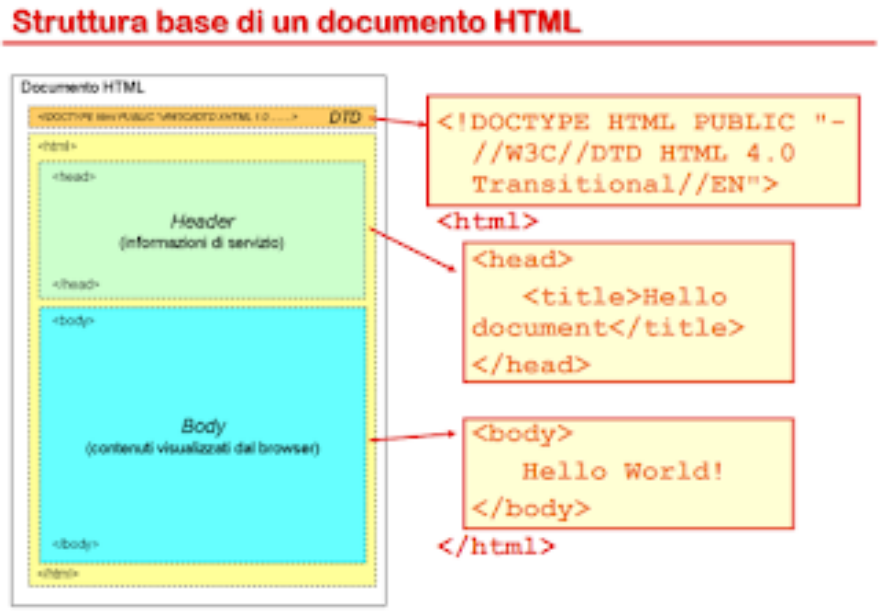
- Lo standard MIME (Multipurpose Internet Mail Extensions) è nato originariamente per poter allegare data file (audio, video, immagini, ...) ai messaggi di posta elettronica
- Oggi, noto anche come Internet Media Type, rappresenta il tipo di contenuto di un messaggio (es. HTTP request)
- Classifica i tipi di contenuto sulla base di una logica a due livelli ed è largamente utilizzata in ambito di HTML e delle tecnologie web in generale
- Un tipo MIME è espresso con questa sintassi: tipo/sottotipo
  - esempi:
    - text/plain: testo semplice
    - text/html: testo HTML

Commenti

- È possibile inserire commenti in qualunque punto all'interno di una pagina HTML con la seguente

sintassi:

<!-- Questo è un testo di commento -->



DTD

- Il primo elemento di un documento HTML è la definizione del tipo di documento (Document Type Definition o DTD)
- Serve al browser per identificare le regole di interpretazione e visualizzazione da applicare al documento
- È costituita da diverse parti:
    - HTML il tipo di linguaggio utilizzato è l'HTML
    - PUBLIC il documento è pubblico
    - - Le specifiche non sono registrate all'ISO (altrimenti +)
    - W3C ente che ha rilasciato le specifiche
    - DTD HTML 4.01 Transitional: versione di HTML
    - EN la lingua con cui è scritta il DTD è l'inglese
    - http://... URL delle specifiche

Header

- È identificato dal tag <head>
- Contiene elementi non visualizzati dal browser (informazioni di servizio)
- <title> titolo della pagina (viene mostrato nella testata della finestra principale del browser)
- <meta> metadati informazioni utili ad applicazioni esterne (es. motori di ricerca) o al browser (es. lingua, codifica dei caratteri utile per la visualizzazione di alfabeti non latini)
- <base> definisce come vengono gestiti i riferimenti relativi nei link
- <link> collegamenti verso file esterni: CSS, script, icone visualizzabili nella barra degli indirizzi del browser
- <script> codice eseguibile utilizzato dal documento
- <style> informazioni di stile (CSS locali)

Elementi <meta>

- Gli elementi di tipo sono caratterizzati da una serie di attributi <meta>
- Esistono due tipi di elementi meta, distinguibili dal primo attributo: http-equiv o name
- Gli elementi di tipo http-equiv danno informazioni al browser su come gestire la pagina e hanno

una struttura di questo tipo: <meta http-equiv=nome content=valore>
    - refresh: indica che la pagina deve essere ricaricata dopo un numero di secondi definito dall'attributo content
    - expires: stabilisce una data scadenza (fine validità) per il documento
    - content type: definisce il tipo di dati contenuto nella pagina (di solito il tipo MIME text/html)
- Gli elementi di tipo name forniscono informazioni utili ma non critiche, hanno una struttura di questo tipo:

<meta name=nome content=valore>

    - author: autore della pagina
    - description: descrizione della pagina
    - copyright: indica che la pagina è protetta da un diritto d'autore
    - keywords: lista di parole chiave separate da virgole, usate dai motori di ricerca
    - date: data di creazione del documento

Body

- Il tag delimita il corpo del documento
- Contiene la parte che viene mostrata dal browser
- Ammette diversi attributi tra cui:
    - background = uri → Definisce l'URI di una immagine da usare come sfondo per la pagina
    - Text = color → Definisce il colore del testo
    - bgcolor = color → In alternativa a background definisce il colore di sfondo della pagina
    - lang = linguaggio → definisce il linguaggio utilizzato nella pagina es. language="it"
- Tipi di elementi del body:
    - Intestazioni: titoli organizzati in gerarchia
    - Strutture di testo: paragrafi, testo indentato, ecc.
    - Aspetto del testo: grassetto, corsivo, ecc.
    - Elenchi e liste: numerati, puntati
    - Tabelle
    - Form (moduli elettronici): campi di inserimento, checkbox e radio button, menu a tendina, bottoni, ecc.
    - Collegamenti ipertestuali e ancore
    - Immagini e contenuti multimediali (audio, video, animazioni, ecc.)
    - Contenuti interattivi: script, applicazioni esterne
- Dal punto di vista del layout della pagina gli elementi HTML si dividono in 3 grandi categorie:
    - Elementi "block-level": costituiscono un blocco attorno a sé, e di conseguenza «vanno a capo» (paragrafi, tabelle, form…)
    - Elementi "inline": non vanno a capo e possono essere integrati nel testo (link, immagini,…)
    - Liste: numerate, puntate

- Regole di composizione:
    - Un elemento block-level può contenere altri elementi dello stesso tipo o di tipo inline
    - Un elemento inline può contenere solo altri elementi inline

Elementi rimpiazzati e non rimpiazzati

- Un'altra distinzione da ricordare è quella tra elementi rimpiazzati (replaced elements) ed elementi non rimpiazzati
    - Gli elementi rimpiazzati sono quelli di cui il browser conosce le dimensioni intrinseche
    - Sono quelli in cui altezza e larghezza sono definite dall'elemento stesso e non da ciò che lo circonda

lo circonda
    - L'esempio più tipico di elemento rimpiazzato è <img>
    - Altri elementi rimpiazzati sono: <input>,<textarea>,<select>
    - Tutti gli altri elementi sono in genere considerati non rimpiazzati

Heading

- I tag <h1>,< h2> ... <h6> servono per definire dei titoli di importanza decrescente (<h1> è il più importante)
- La "h" sta per "heading", cioè titolo e sono previste 6 grandezze; i titoli appaiono in grassetto e lasciano una riga vuota prima e dopo di sé (sono elementi di blocco)
- Ammettono attributi di allineamento: <h1 align = left | center | right | justify>

Paragrafi

- Il paragrafo è l'unità di base entro cui suddividere un testo: è un elemento di tipo blocco
- Il tag lascia una riga vuota prima della sua apertura e dopo la sua chiusura

- Se si vuole andare a capo all'interno di un paragrafo si usa l'elemento

Div

- Se al posto di <p> si usa il tag il blocco di testo va a capo, ma - a differenza del paragrafo - non lascia spazi prima e dopo la sua apertura
- ! È l'elemento di tipo blocco per eccellenza

Span

- Lo è un contenitore generico che può essere annidato (ad esempio) all'interno dei
- È un elemento inline, e quindi non va a capo ma continua sulla stessa linea del tag che lo include
- È un elemento molto utilizzato soprattutto insieme ai fogli di stile per dare un aspetto particolare ad un pezzo di testo in un blocco (per esempio, per evidenziare)
- ! Se non viene associato ad uno stile è invisibile

Horizontal rule

- Il tag <hr> serve ad inserire una riga di separazione
- Attributi:
    - align = {left | center | right} ⟶ Allineamento della riga rispetto a ciò che la circonda
    - size = pixels ⟶ Altezza della riga
    - width = length ⟶ Larghezza della riga in modo assoluto o in percentuale delle dimensioni di ciò che la contiene
    - noshade ⟶ Riga senza effetto di ombreggiatura

Stili del testo

- Nella terminologia tipografica lo "stile di un testo" indica le possibili varianti di forma di un carattere: tondo (normale), neretto (grassetto), corsivo
- HTML consente di definire lo stile di un frammento di testo, combinando fra loro anche più stili
- I tag che svolgono questa funzione vengono normalmente suddivisi in fisici e logici:
    - Tag fisici: definiscono lo stile del carattere in termini grafici indipendentemente dalla funzione nel documento
        - es: <tt>...</tt>, ..., ..., <u>...</u>, <s>...</s>
    - Tag logici: forniscono informazioni sul ruolo svolto dal contenuto, e in base a questo adottano uno stile grafico
        - es: resa (enfasi) <code> (prog...

- es: , (emphasis), <code>/<pre>, <kbd>, <abbr>, <acronym>, <address>, <blockquote>, <cite>

Font

- Il tag permette di formattare il testo, definendo dimensioni, colore, tipo di carattere
- È l'esempio limite del mescolamento fra contenuto e rappresentazione
- È deprecato in HTML 4.01
- Attributi:
    - size = [+ | -] n → Definisce le dimensioni del testo (1-7 o relative)
    - color = color → Definisce il colore del testo
    - face = text → Definisce il font del testo

Liste

- Il tag <ul> (unordered list) permette di definire liste non ordinate (puntate)
- Gli elementi della lista vengono definiti mediante il tag <li> (list item)
- L'attributo type definisce la forma dei punti e ammette 3 valori: disc, circle e square
- Il tag <ul> (unordered list) permette di definire liste non ordinate (puntate)
- Gli elementi della lista vengono definiti mediante il tag <li> (list item)
- L'attributo type definisce la forma dei punti e ammette 3 valori: disc, circle e square
- Il tag <dl> (definition list) permette di definire liste di definizione
- Sono liste costituite alternativamente da termini (tag <dt>) e definizioni (tag <dd>)

Tabelle

- Il tag <table> racchiude la tabella
- Attributi:
    - align = "{left|center|right}" → allineamento della tabella rispetto alla pagina;
    - width="n|n%" → larghezza della tabella (anche in percentuale rispetto alla pagina);
    - bgcolor="#xxxxxx" → colore di sfondo della tabella;
    - border="n" → spessore dei bordi della tabella (0 = tabella senza bordi);
    - cellspacing, cellpadding
- <tr> è il tag che racchiude ciascuna riga della tabella
- Attributi:
    - align = "{left|center|right|justify}" → allineamento del contenuto delle celle della riga;
    - valign = "{top|middle|bottom|baseline}" → allineamento verticale del contenuto delle celle della riga;
    - bgcolor="#xxxxxx" → colore di sfondo della riga
- <th> e <td> sono i tag che racchiudono le celle
    - <th> serve per le celle della testata
    - <td> serve per le celle del contenuto
- Attributi:
    - Gli stessi di <tr>
    - width, height = {length|length%} → specifica le dimensioni (larghezza e altezza) della cella, dimensione assoluta (pixels) o valore percentuale;
    - rowspan, colspan = n → indica su quante righe/colonne della tabella si estende la cella
- ■ Le tabelle permettono di costruire griglie in cui inserire i contenuti di un sito e per mezzo degli sfondi e dei margini è possibile riprodurre un'impostazione accattivante
- Permettono di realizzare i cosiddetti layout "liquidi", che si adattano cioè alla risoluzione del monitor dell'utente (grazie all'uso delle dimensioni in %)
- La tendenza attuale è quella di superare questa tecnica, che presenta alcuni inconvenienti:
    - Mischia elementi di formattazione dei dati ai dati stessi
    - Appesantisce le pagine con molti elementi, rallentando lo scaricamento

    - Appesantisce le pagine con molti elementi, rallentando lo scaricamento
- ! Siamo comunque in una fase di transizione e l'impaginazione a tabelle è ancora molto, molto usata

Link ipertestuali

- Il link è il costrutto di base di un ipertesto
- Caratterizza HTML come linguaggio a marcatori per la descrizione di ipertesti
- È una connessione fra una risorsa Web ed un'altra
- Un link è costituito da due estremi - detti àncore (anchor) - e da una direzione di percorrenza
    - Link = source anchor → destination anchor
    - L'àncora di origine (source anchor) è un elemento contenuto nella pagina di partenza
    - L'àncora di destinazione (destination anchor) è una qualsiasi risorsa web (un'immagine, un video, un eseguibile, un documento HTML o un elemento interno al documento)
    - La risorsa di destinazione si ottiene «visitando» il link

Ancore

- In HTML le ancore, sia di origine che di destinazione, si esprimono utilizzando il tag <a>
- Le àncore di origine sono caratterizzate da un attributo, denominato href, che contiene l'indirizzo di destinazione (è un URL)
- Le àncore di destinazione sono invece caratterizzate dall'attributo name
- L'esempio più semplice di link è quello che collega due elementi all'interno di uno stesso documento
    - In questo caso l'attributo href dell'àncora di origine
    - ha la forma #nome
    - nome è il valore dell'attributo name dell'àncora di destinazione
    - Un elemento #xxxx posto alla fine di un URL viene chiamato fragment
- Si può esprimere un'àncora di destinazione in forma "implicita", cioè senza utilizzare il tag <a>
- È sufficiente assegnare l'attributo ID a un qualunque elemento della pagina
- È una forma più compatta anche se probabilmente meno facile da interpretare
- Caso più comune = link ad un altro documento (pagina HTML) o in generale ad un'altra risorsa (es. un'immagine)
- Se il link non specifica un'àncora si "salta" all'inizio del documento specificato
- Gli URL utilizzati nell'attributo HREF possono essere assoluti o relativi; se relativi si procede alla «risoluzione» utilizzando come base quella del documento corrente. Se desidero comportamento diverso, attivo l'attributo <base> dell'header
- cosa succede quando clicco su un'ancora?
    1. L'URL definito dall'attributo HREF viene «risolto»
    2. Se è un URL HTTP, viene fatta una chiamata HTTP al server in cui si trova il documento; chiamata di tipo GET per ottenere la risorsa descritta dall'URL
    3. La pagina viene caricata e visualizzata dal browser
    4. Se è stata definita anche la parte fragment (#xxxxxx) il browser si porta al punto della pagina specificato

Immagini

- Il tag <img> consente di inserire immagini in un documento HTML con la sintassi: <img src ="sitemap.gif">
- Attributi:
    - src = uri → specifica l'indirizzo dell'immagine (required)
    - alt = text → testo alternativo nel caso fosse impossibile visualizzare l'immagine
    - align = {bottom|middle|top|left|right} (deprecato in HTML 4.01) → posizione rispetto al testo che la circonda
    - width,height = pixels → larghezza e altezza dell'immagine in pixel

- width,height = pixels → larghezza e altezza dell'immagine in pixel
- border = pixels (deprecato in HTML 4.01) → spessore del bordo dell'immagine (0 = nessun bordo)

Form

- Un form (modulo) è una sezione di documento HTML che contiene elementi di controllo che l'utente può utilizzare per inserire dati o in generale per interagire
- I dati inseriti possono essere poi inoltrati al server dove un agente può processarli
- Gli elementi di controllo sono caratterizzati da un valore iniziale e da un valore corrente
- Gli elementi di controllo possono essere:
  - Bottoni di azione
  - Checkbox (caselle di spunta)
  - Radio Button (bottoni mutuamente esclusivi)
  - Liste di selezione (lista di opzioni)
  - Caselle di inserimento di testo
  - Oggetti nascosti (elementi valorizzati ma invisibili)
- Il tag <form> racchiude tutti gli elementi del modulo (è un elemento di tipo blocco)
- Attributi:
  - action = uri → URI dell'agente che riceverà i dati del form
  - name = text → specifica il nome del form
  - method = {get|post} → specifica il modo in cui i dati vengono inviati
  - enctype = content-type → se il metodo è POST specifica il content type usato per la codifica (encoding) dei dati contenuti nel form; default application/x-www-form-urlencoded
- La maggior parte dei controlli viene definita mediante il tag <input>, l'attributo type stabilisce il tipo di controllo
  - text: casella di testo monoriga
  - password: come text ma il testo non è leggibile (****)
  - file: controllo che consente di caricare un file
  - checkbox: casella di spunta
  - radio: radio button
  - submit: bottone per trasmettere il contenuto del form
  - image: bottone di submit sotto forma di immagine
  - reset: bottone che riporta tutti i campi al valore iniziale
  - button: bottone di azione
  - hidden: campo nascosto
- Tutti gli input possono essere disabilitati utilizzando l'attributo disabled nella forma disabled = "disabled"

DOM

- Una pagina HTML può essere rappresentata come una struttura ad albero
- Questa struttura logica prende il nome di DOM: Document Object Model
- Quando un browser riceve una pagina HTML, ne fa il parsing e costruisce la struttura ad albero del DOM

HTML 5

- molto di ciò che il linguaggio HTML 5 offre rispetto a HTML 4.01 è opzionale: possiamo dunque utilizzare il linguaggio HTML 5 programmando con costrutti HTML 4.01 e «attivando» le sole «feature» HTML 5 di cui abbiamo veramente bisogno
- Questa ortogonalità dei «nuovi costrutti HTML 5» favorisce la proliferazione di documenti HTML 5 «100%»

→ nuova formula semplificata del doctype <!DOCTYPE HTML>

→ nuova formula semplificata del doctype <!DOCTYPE HTML>
...in cui però la stragrande maggioranza della sintassi è puro HTML 4.01!

- Tra le tante cose, HTML 5 estende notevolmente la possibilità di integrazione di contenuti multimediali nella pagina

→ contenuti con i quali è possibile interagire in modo avanzato

- Il modello ad eventi di DOM è esteso con eventi specifici che permettono la costruzione di applicazioni sofisticate client-side. Agli eventi si aggiungono nuove API per la manipolazione degli oggetti DOM
    - **<canvas>** : L'elemento <canvas> definisce un'area rettangolare in cui disegnare direttamente immagini bidimensionali e modificarle in relazione a eventi, tramite funzioni Javascript
    - **<audio>, <video>** : meccanismo generico per il caricamento di file e stream video o audio, più alcune proprietà DOM per controllarne l'esecuzione. Può contenere diversi elementi <source> che specificano diversi file, tra i quali il browser sceglie quello da eseguire
        - I membri WHATWG non sono riusciti a trovare un accordo sul formato di codec da usare per i video
        - La «soluzione pratica» ad oggi è non indicare un formato nelle specifiche HTML e permettere l'uso di diversi codec
- HTML 5 introduce anche utili API specifiche per servizi avanzati di geo-localizzazione:
    - getCurrentPosition(), restituisce latitudine e longitudine della posizione dell'utente
    - watchPosition(), restituisce la posizione corrente dell'utente e continua ad aggiornare la posizione dello stesso durante i suoi spostamenti (come in una applicazione di navigazione GPS)
    - ClearWatch(), termina il metodo watchPosition()

CSS

- I fogli di stile a cascata (Cascading Style Sheets = CSS) hanno lo scopo fondamentale di separare contenuto e presentazione nelle pagine Web
    - Il linguaggio HTML serve a definire il contenuto (! ma anche struttura, linking e semantica) senza tentare di dare indicazioni su come rappresentarlo
    - I CSS servono a definire come il contenuto deve essere presentato all'utente
- La parola chiave del linguaggio CSS è cascading:
    - È prevista ed è incoraggiata la presenza di fogli di stile multipli, che agiscono uno dopo l'altro, in cascata, per indicare le caratteristiche tipografiche e di layout di un documento HTML
- I vantaggi della separazione di competenze sono evidenti:
    - Lo stesso contenuto diventa riusabile in più contesti
    - Basta cambiare i CSS e può essere presentato correttamente in modo ottimale su dispositivi diversi (es. PC e palmari) o addirittura su media diversi (es. video e carta)
    - Si può dividere il lavoro fra chi gestisce il contenuto e chi si occupa della parte grafica
- Tra gli altri obiettivi:
    - Ridurre i tempi di scaricamento delle pagine: una pagina che usa i CSS è meno della metà di una pagina che usa la formattazione con tag HTML, inoltre se il file CSS è condiviso da più pagine viene scaricato una volta sola (caching CSS come risorsa statica)
    - Ripulire il codice HTML: eliminare il proliferare e l'uso di estensioni
    - Rendere le pagine «visualizzabili» e «usabili» da dispositivi non convenzionali: laptop, palmari, smartphone, ecc.
- Tra le caratteristiche principali dei CSS:
    - Il controllo sia dell'autore sia del lettore di un documento HTML

- Il fatto di essere indipendenti dalla specifica collezione di elementi ed attributi HTML, così da rendere possibile e facile il supporto di nuove versioni di HTML e anche di XML

- Un documento HTML può essere visto come un insieme di blocchi (contenitori) sui quali si può agire con stili diversi; ogni tag HTML definisce un blocco e si può identificare assegnando un id o una class
- Riferire un css esterno:

  <LINK rel="stylesheet" href="hello.css" type="text/css">

  oppure

  <style type="text/css"> @import url(hello.css); </style>
  - Quella con <link> è più diffusa
  - Quella con @import è meno critica per la compatibilità con i vecchi browser
- È sicuramente preferibile usare gli stili esterni:
  - È facile applicare diversi stili alla stessa pagina
  - Si ottimizza il trasferimento delle pagine perché il foglio stile rimane nella cache del browser
- L'uso degli stili inline è da evitare
  - Rendono molto basso il livello di separazione tra contenuto e presentazione
  - Le modifiche sono molto complicate
- Gli stili interni sono una via di mezzo

Regole CSS e selettori

- Un'espressione come H1 { color: blue } prende il nome di regola CSS ed è composta da due parti:
  - Selettore: H1
  - Dichiarazione: color: blue → a sua volta si divide in due parti:
    - Proprietà: color
    - Valore: blue

- Il selettore serve per collegare uno stile agli elementi a cui deve essere applicato
- Selettore universale: identifica qualunque elemento

  `* { … }`
- Selettori di tipo: si applicano a tutti gli elementi di un determinato tipo (ad es. tutti i <p>)

  `tipo_elemento { … }`
- Classi: si applicano a tutti gli elementi che presentano l'attributo `class="nome_classe"`

  `.nome_classe { … }`
- Identificatori: si applicano agli elementi che presentano l'attributo `id="nome_id"`

  `#nome_id { … }`

- La sintassi generale si può quindi esprimere così
  - selettore { proprietà: valore }
- O più in generale:
  - selettore {
  - proprietà1 : valore1;
  - proprietà2 : valore2, valore3; }
- I selettori di tipo si possono combinare con quelli di classe e di identificatore:
  - tipo_elemento.nome_classe { … }
  - tipo_elemento#nome_id { … }
- Pseudoclassi: si applicano ad un sottoinsieme degli elementi di un tipo identificato da una proprietà
  - tipo_elemento:proprietà { … }
  - Es. stato di un'ancora: link e visited → a:link { … }, a:visited { … }
  - Es. condizione di un elemento: active, focus e hover → h1:hover { … },
- Pseudoelementi: si applicano ad una parte di un elemento

- tipo_elemento:parte { … }
- Es. solo la prima linea o la prima lettera di un paragrafo:
  - p:first-line { … }
  - p:first-letter { … }

- Selettori gerarchici: si applicano a tutti gli elementi di un dato tipo che hanno un determinato legame gerarchico (discendente, figlio, fratello) con elementi di un altro tipo
  - tipo1 tipo2 { … } tipo2 discende da tipo1
  - tipo1>tipo2 { … } tipo2 è figlio di tipo1
  - tipo1+tipo2 { … } tipo2 è fratello di tipo1
- Ad esempio: UL>LI { … } si applica solo aqli elementi contenuti direttamente in liste non ordinate

Se la stessa dichiarazione si applica a più tipi di elemento si scrive una regola in forma raggruppata

```
H1 { font-family: sans-serif }
H2 { font-family: sans-serif }
H3 { font-family: sans-serif }
```

equivale a

```
H1, H2, H3 { font-family: sans-serif }
```

Proprietà

- Nelle dichiarazioni è possibile far uso di proprietà singole o in forma abbreviata (shorthand properties)
  - Le proprietà singole permettono di definire un singolo aspetto di stile
  - Le shorthand properties consentono invece di definire un insieme di aspetti, correlati fra di loro usando una sola proprietà
- Per esempio, ogni elemento permette di definire un margine rispetto a quelli adiacenti usando quattro proprietà: margin-top, margin-right, margin-bottom, margin-left
- Utilizziamo le proprietà singole applicandole ad un paragrafo:
  P { margin-top: 10px; margin-right: 8px; margin-bottom: 10px; margin-left: 8px; }
- Lo stesso risultato si può ottenere usando la proprietà in forma abbreviata margin:
  P {margin: 10px 8px 10px 8px;}

Valori - 1

- Numeri interi e reali: "." come separatore decimale
- Grandezze: usate per lunghezze orizzontali e verticali
  - un numero seguito da una unità di misura
- Unità di misura relative
  - em: è relativa alla dimensione del font in uso (es. se il font ha corpo 12pt, 1em varrà 12pt, 2em varranno 24pt)
  - px: pixel, sono relativi al dispositivo di output e alle impostazioni del computer dell'utente
- Unità di misura assolute
  - in: pollici; (1 in =2.54 cm)
  - cm: centimetri
  - mm: millimetri
  - pt: punti tipografici (1/72 di pollice)
  - pc: pica = 12 punti

Valori - 2

- **Percentuali**: percentuale del valore che assume la proprietà stessa nell'elemento padre; un numero seguito da %
- **URL** assoluti o relativi; si usa la notazione `url(percorso)`
- **Stringhe**: testo delimitato da apici singoli o doppi
- **Colori**: possono essere identificati con tre metodi differenti:
  - In forma esadecimale `#RRGGBB`
  - Tramite la funzione `rgb(rosso,verde,blu)`
  - Usando una serie di **parole chiave** che possono indicare colori assoluti o dipendenti dall'impostazione del PC (proprietà di sistema)

Ereditarietà

- Per poter rappresentare una pagina HTML il browser deve riuscire ad applicare ad ogni elemento uno stile
- Un elemento privo di stile non può essere rappresentato: anche se nella pagina non c'è nessuna regola CSS (interna o esterna) ogni browser ha un foglio stile di default che contiene stili per ogni tipologia di elemento HTML (tag)
- L'attribuzione può essere diretta:
  - L'elemento contiene uno stile inline
  - Esistono una o più regole il cui selettore rimanda all'elemento
- Oppure può essere indiretta:
  - L'elemento "eredita" lo stile dall'elemento che lo contiene
- Ereditarietà = meccanismo di tipo differenziale simile per certi aspetti all'ereditarietà nei linguaggi ad oggetti
- Si basa sui blocchi annidati di un documento HTML
- Uno stile applicato ad un blocco esterno si applica anche ai blocchi in esso contenuti
- In un blocco interno:
  - Si possono definire stili aggiuntivi
  - Si possono ridefinire stili definiti in un blocco più esterno (è una sorta di overriding)
- Lo stesso ragionamento si può esprimere in termini di DOM:
  - un nodo figlio eredita gli stili dei nodi che si trovano sul ramo da cui discende
- Non tutte le proprietà sono soggette al meccanismo dell'ereditarietà
- In generale non vengono ereditate quelle che riguardano la formattazione del box model
- Il box è il riquadro che circonda ogni elemento
- La motivazione è abbastanza intuitiva: se ogni elemento interno ereditasse le proprietà dell'elemento che lo contiene avremmo un effetto grafico tipo "scatole cinesi" assolutamente indesiderato

Conflitti di stile

- Nell'applicare gli stili possono nascere conflitti di competenza per diversi motivi:
  - Esiste un'intersezione tra regole che utilizzano selettori di tipo diverso, ad esempio ID e Classe come in questo caso: p#myID {text-color=red} p.myClass {text-color=blue} …
  - Una pagina usa più fogli di stile oppure combina fogli di stile esterni e regole interne o

inline

- Nello stesso foglio stile ci sono regole con lo stesso selettore e dichiarazioni diverse
- banale errore o gestione disordinata dei CSS

- Lo standard definisce un insieme di regole di risoluzione dei conflitti che prende il nome di cascade. Si basa su tre elementi
  - Origine del foglio stile:
    - Autore: stile definito nella pagina
    - Browser: foglio stile predefinito
    - Utente: in alcuni browser si può editare lo stile
  - Specificità: esiste una formula che misura il grado di specificità attribuendo, ad es., un punteggio maggiore ad un selettore di ID rispetto ad uno di Classe
  - Dichiarazione !important: è possibile aggiungere al valore di una dichiarazione la clausola !important p.myClass {text-color: red !important}

- Il CSS assegna un peso a ciascun blocco di regole
- In caso di conflitto vince quella con peso maggiore
- Per determinare il peso si applicano in sequenza una serie di regole:
  - Origine: l'ordine di prevalenza è autore, utente, browser
  - Specificità del selettore: ha la precedenza il selettore con specificità maggiore
  - Ordine di dichiarazione: se esistono due dichiarazioni con ugual specificità e origine vince quella fornita per ultima.
    - N.B. Le dichiarazioni esterne sono considerate come precedenti a qualsiasi dichiarazione interna

- effetto della clausola !important: ha sempre precedenza, indipendentemente da origine, specificità e ordine di apparizione

CSS 3

- I CSS 3 estendono le funzionalità dei CSS 2.1 mantenendone la piena compatibilità
- A differenza di CSS 2.1, che si presentata come unica grande specifica, CSS 3 è diviso in diversi moduli
- Nuovi selettori, pseudo-classi e pseudo-elementi: i CSS 3 si avvalgono ora di selettori più potenti che permettono di selezionare gli elementi in base alla presenza di nuovi token non contemplati in CSS 2.1 e di nuovi contesti del DOM
    - ○ se prima potevamo selezionare un elemento in base alla presenza di un attributo: p[class]{} ora possiamo selezionarlo anche in base alla presenza di particolari stringhe al suo interno: p[class^="html"]{}
    - ○ ancora, possiamo selezionare elementi in base alla loro posizione nel DOM: p:nth-child(3) {} o in base al loro particolare stato nella user interface: input:disabled{}
- Supporto completo a XML: i CSS 2.1 non erano in grado di gestire i namespace XML. I CSS 3 sono in grado di farlo tramite la regola @namespace element "http://www.sito.it/ns/element/"
- I CSS 3 consentono di indirizzare gli stili non solo ai dispositivi specifici, ma anche di specificare quali condizioni devono essere soddisfatte per l'applicazione di tali stili tramite le nuove regole @media screen and (max-width: 800px){}
- Con i CSS 3 è ora possibile creare intestazioni, piè di pagina, note in calce e numerazioni automatiche delle pagine

WEB DINAMICO

- Contenuti composti dinamicamente in base all'interazione con l'utente

CGI

- 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 linquaqqio ibrido (es. Python)

- 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
- 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
- Con il metodo GET il server passa il contenuto della form al programma CGI come se fosse da linea di comando di una shell
    - ○ Linea di comando ha una lunghezza finita dipendente da SO (massimo 256 caratteri su SO UNIX)

    → quantità di dati che possono essere inviati con il metodo GET è molto limitata
- Usando POST non viene aggiunto nulla alla URL specificata da ACTION → linea comando nella shell contiene solo il nome del programma CGI
    - ○ I dati del form, contenuti nell'header HTTP, vengono inviati al programma CGI tramite standard input

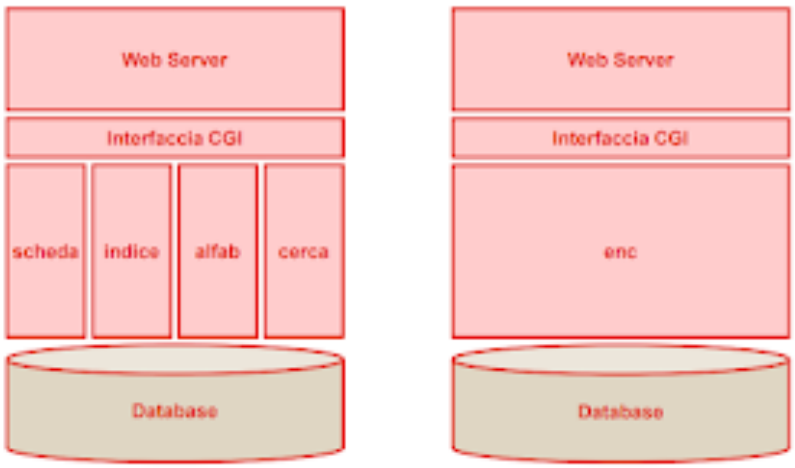
    → In questo modo si possono inviare dati lunghi a piacimento, senza i limiti di GET
- Prima di chiamare il programma CGI, Web server imposta alcune variabili di sistema corrispondenti ai principali header HTTP, ad esempio:
    - ○ REQUEST_METHOD: metodo usato dalla form
    - ○ QUERY_STRING: parte di 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
- 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
- Il server preleva i dati dallo standard output e li invia al client incapsulandoli in messaggio HTTP
- server deve rendersi conto che 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 dove 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
- Si può creare programma CGI che genera dinamicamente pagine basandosi su database
- 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)
- 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
- Per ovviare al penultimo punto si potrebbe realizzare una sola CGI (enc) che implementa tutte le funzionalità
- In questo modo però si ha un'applicazione monolitica e si perdono i vantaggi della modularità
- Gli altri problemi rimangono invariati

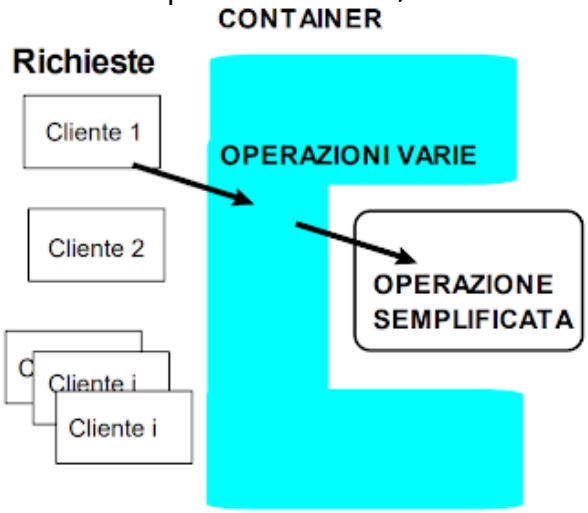
Le due soluzioni a confronto



- 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

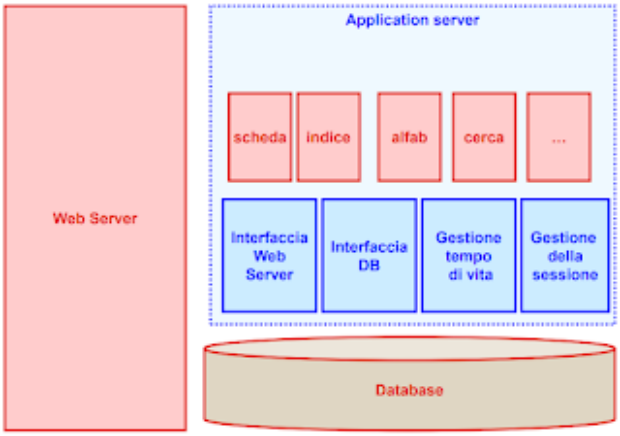
Modelli a contenimento

- 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 e 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
- Un servizio utente potrebbe essere integrato in un ambiente (middleware) che si occupa in modo autonomo di molti aspetti diversi
- Container possono ospitare componenti più trasportabili, riutilizzabili e mobili
- Il container può fornire "automaticamente" molte delle funzioni per supportare il servizio applicativo verso l'utente, es:
    - 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
    - ...

Interazione stateful e stateless

- L'interazione tra un Client e un Server può essere di due tipi:
    - Stateful: esiste stato dell'interazione, 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
- 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
    - In generale, molto spesso abbiamo a che fare con operazioni idempotenti nelle applicazioni Web
- 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

- 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
- 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)

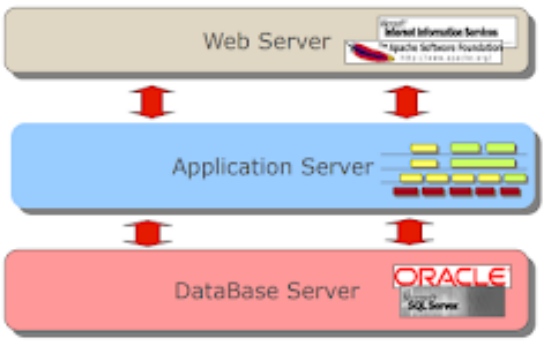
Esempio di conversazione: acquisto online

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

- 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

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)

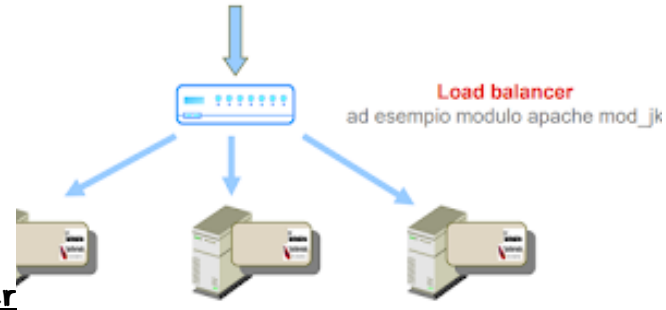


Distribuzione dei servizi

- 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
- 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
- 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

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

Applicazione

- Stato di sessione prevalentemente

● 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 completamente replicabile in modo orizzontale.

Database

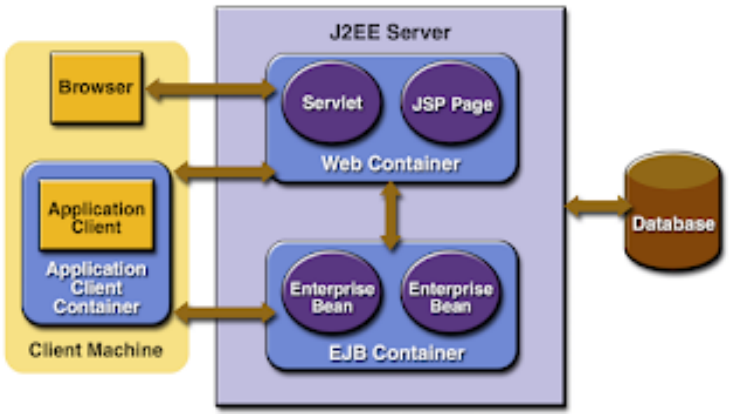
● Il database server è (normalmente) un server stateful. → Perché? Con quali problemi conseguenti?
● La replicazione è molto delicata perché deve mantenere il principio di atomicità delle transazioni
● I database commerciali, come Oracle e Microsoft SQL Server prevedono delle configurazioni di clustering (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)

SERVLET

● Modello a Componente-Container → architetture multi-tier, servizi ad alta scalabilità per applicazioni distribuite enterprise, anche Web-based

● I Web Client hanno sostituito, in molte situazioni di applicazioni client-server, i più tradizionali "fat client"

  ○ sono spesso costituiti dal semplice browser Web senza bisogno di alcuna installazione ad



hoc

  ○ comunicano via HTTP e HTTPS con il server

  ○ effettuano il rendering della pagina in HTML (o altre tecnologie mark-up)

  ○ possono essere sviluppati utilizzando varie tecnologie

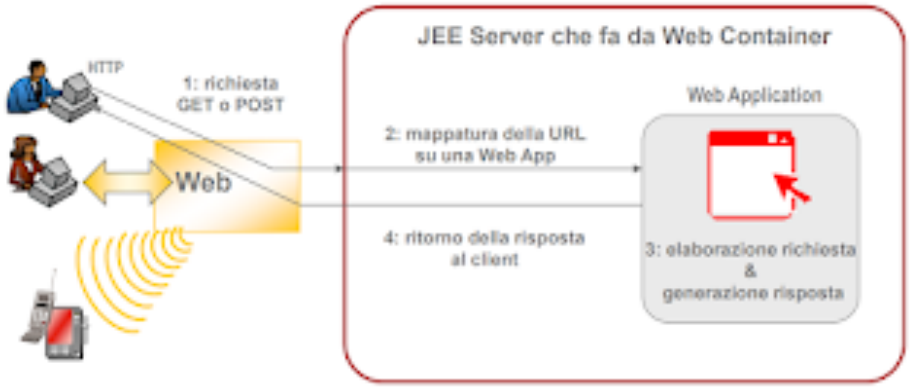
  ○ sono spesso implementati come parti di architetture multi-tier

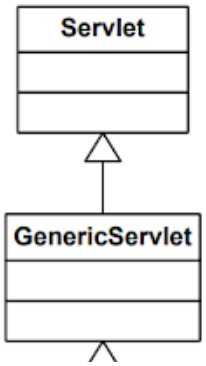
● Una Web Application è un gruppo di risorse server side che nel loro insieme creano una

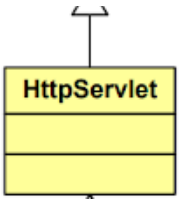
applicazione interattiva fruibile via Web

- Le risorse server-side includono:
  - Classi server-side (Servlet e classi standard Java)
  - Java Server Pages (le vedremo in seguito)
  - Risorse statiche (documenti HTML, immagini, css, ...)
  - Applet, Javascript e/o altri componenti che diventeranno attivi client-side
  - Informazioni di configurazione e deployment
- I Web Container forniscono un ambiente di esecuzione per Web Application
- In generale, Container garantiscono servizi di base alle applicazioni sviluppate secondo un paradigma a componenti

Accesso a una Web Application:







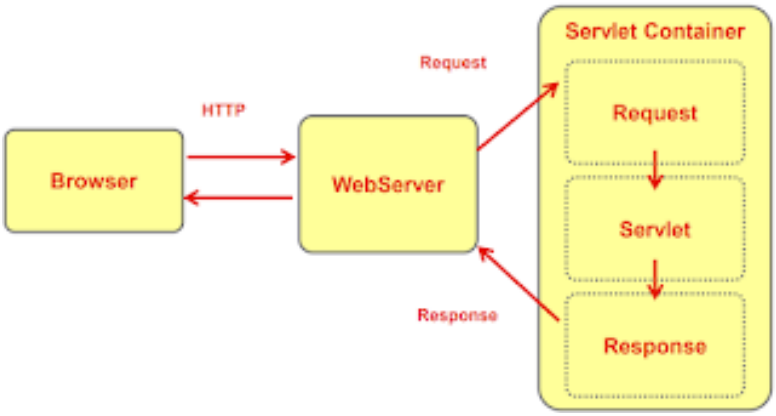
Che cos'è una Servlet?

- Una Servlet è una classe Java che fornisce risposte a richieste HTTP seguendo un protocollo condiviso; in termini più generali è una classe che fornisce un servizio comunicando con il client mediante protocolli di tipo request/response: tra questi protocolli il più noto e diffuso è HTTP
- Le Servlet estendono le funzionalità di un Web server generando contenuti dinamici e superando i classici limiti delle applicazioni CGI
- Eseguono direttamente in un Web Container
    - In termini pratici sono classi che derivano dalla classe HttpServlet
    - HttpServlet implementa vari metodi che possiamo ridefinire

Modello request-response

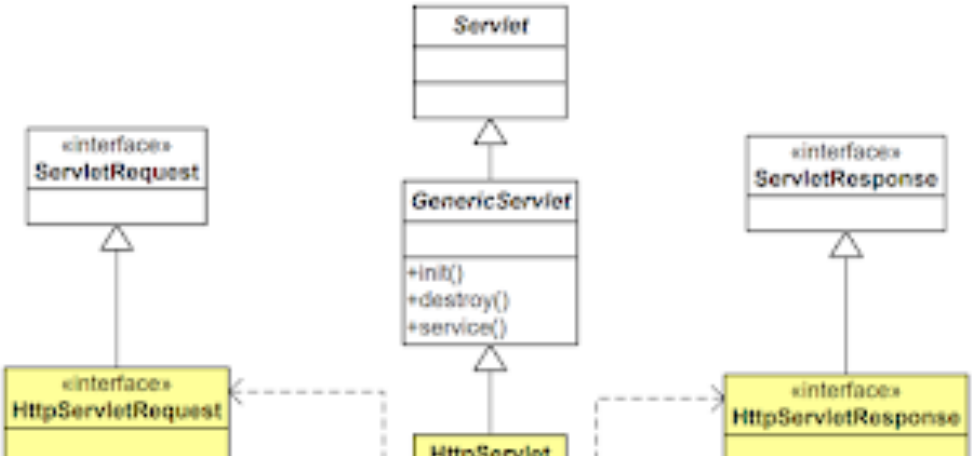
- All'arrivo di una richiesta HTTP il Servlet Container crea un oggetto request e un oggetto response e li passa alla servlet
- Gli oggetti di tipo Request rappresentano la chiamata al server effettuata dal client

→ caratterizzati da varie informazioni:



- Chi ha effettuato la Request
- Parametri passati nella Request
- Header passati

- Gli oggetti di tipo Response rappresentano le informazioni restituite al client in risposta ad una Request:
    - Dati in forma testuale (es. html, text) o binaria (es. immagini)
    - HTTP header, cookie, ...



Ciclo di vita delle Servlet

- Servlet container controlla e supporta automaticamente il ciclo di vita di una servlet
- Se non esiste una istanza della servlet nel container:
  - Carica la classe della servlet
  - Crea un'istanza della servlet
  - Inizializza la servlet (invoca il metodo init())
- Poi, a regime:
  - Invoca la servlet (doGet() o doPost() a seconda del tipo di richiesta ricevuta) passando come parametri due oggetti di tipo HttpServletRequest e HttpServletResponse

→ Quante istanze di servlet? Quanti thread sono associati ad una istanza di servlet? Quale modello di concorrenza? Con quali pericoli?

- Modello "normale": una sola istanza di servlet e un thread assegnato ad ogni richiesta http per servlet, anche se richieste per quella servlet sono già in esecuzione

→ Nella modalità normale più thread condividono la stessa istanza di una servlet e quindi si crea una situazione di concorrenza

  - Il metodo init() della servlet viene chiamato una sola volta quando la servlet è caricata dal Web container
  - I metodi service() e destroy() possono essere chiamati solo dopo il completamento dell'esecuzione di init()
  - Il metodo service() (e quindi doGet() e doPost()) può essere invocato da numerosi client in modo concorrente ed è quindi necessario gestire le sezioni critiche (a completo carico del programmatore dell'applicazione Web)

  → Uso di blocchi synchronized / semafori / mutex

- Alternativamente si può indicare al container di creare un'istanza della servlet per ogni richiesta concorrente

→ Single-Threaded Model

  - È onerosa in termine di risorse ed è deprecata nelle specifiche 2.4 delle servlet
  - Se una servlet vuole operare in modo single-threaded deve implementare l'interfaccia marker SingleThreadModel (altro esempio di interfaccia marker in java è Serializable)

Metodi per il controllo del ciclo di vita:

- init(): viene chiamato una sola volta al caricamento della servlet
  - In questo metodo si può inizializzare l'istanza: ad esempio si crea la connessione con un database
- service(): viene chiamato ad ogni HTTP Request
  - Chiama doGet() o doPost() a seconda del tipo di HTTP Request ricevuta
  - metodo astratto
  - HTTPServlet fornisce una implementazione di service() che delega l'elaborazione della richiesta ai metodi doGet(), doPost(), doPut() e doDelete()
- destroy(): viene chiamato una sola volta quando la servlet deve essere disattivata (es. quando è rimossa)
  - Tipicamente serve per rilasciare le risorse acquisite (es. connessione a db, eliminazione

di variabili di stato per l'intera applicazione, ...)
- definiti nella classe astratta GenericServlet

Oggetto response

- Contiene i dati restituiti dalla Servlet al Client:
  - Status line (status code, status phrase). Per definire lo status code HttpServletResponse fornisce il metodo public void *setStatus(int statusCode)*
    → Per inviare errori possiamo anche usare *public void sendError(int sc) / public void sendError(int code, String message)*
  - Header della risposta HTTP → metodi *setHeader*, *addHeader*, ***setContentType***, *addCookie*, ...
  - Response body: il contenuto (ad es. pagina HTML)
  - Ha come tipo l'interfaccia HttpResponse che espone metodi per:
    - Specificare lo status code della risposta HTTP
    - Indicare il content type (tipicamente text/html)
    - Ottenere un output stream in cui scrivere il contenuto da restituire
    - Indicare se l'output è bufferizzato
    - Gestire i cookie
    - ...

In rosso ciò che può/deve essere specificato a livello di codice della servlet

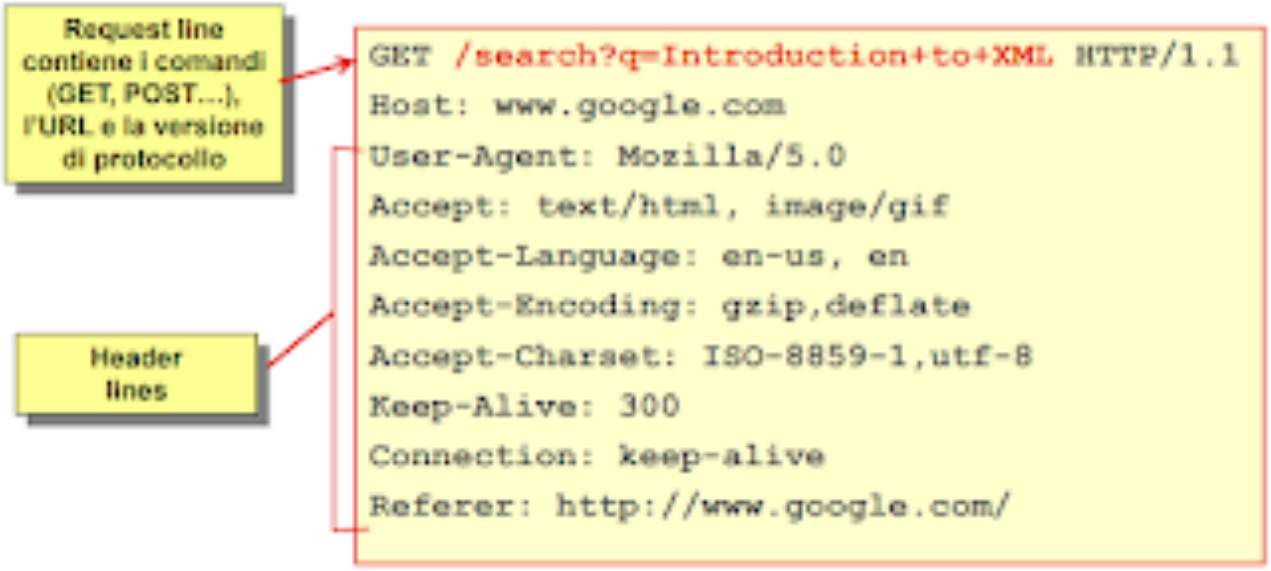
Gestione del contenuto:

- Per definire il response body possiamo operare in due modi utilizzando due metodi di response
- *public PrintWriter getWriter*: mette a disposizione uno stream di caratteri (un'istanza di PrintWriter)
  - utile per restituire un testo nella risposta (tipicamente HTML)
- *public ServletOutputStream getOuputStream()*: mette a disposizione uno stream di byte (un'istanza di ServletOutputStream)
  - più utile per una risposta con contenuto binario (per esempio un'immagine)

request

- Ricordiamo che in un URL (e quindi in una GET)possiamo inserire una query string che ci permette di passare parametri con la sintassi: <path>?<nome1>=<valore1>&<nome2>=<valore2>&...
- Per ricavare il parametro utilizzeremo il parametro request passato a doGet()

- request contiene i dati inviati dal client HTTP al server
- Viene creata dal servlet container e passata alla servlet come parametro ai metodi doGet() e doPost()
- È un'istanza di una classe che implementa l'interfaccia HttpServletRequest
- Fornisce metodi per accedere a varie informazioni:
    - HTTP Request URL
    - HTTP Request header
    - Tipo di autenticazione e informazioni su utente
    - Cookie
    - Session (lo vedremo nel dettaglio in seguito)



- Come sapete bene, una URL HTTP ha la sintassi http://[host]:[port]/[request path]?[query string]
- La request path è composta dal contesto e dal nome della Web application
- La query string è composta da un insieme di parametri che sono forniti dall'utente
- Non solo da compilazione form; può apparire in una pagina Web in un anchor: Add To Cart
- Il metodo getParameter() di request ci permette di accedere ai vari parametri
    - Ad esempio se scriviamo: String bookId = request.getParameter("Add"); bookID varrà "101"

Metodi per accedere all'URL

- String getParameter(String parName) → restituisce il valore di un parametro individuato per nome
- String getContextPath() → restituisce informazioni sulla parte dell'URL che indica il contesto della Web application
- String getQueryString() → restituisce la stringa di query
- String getPathInfo() → per ottenere il path
- String getPathTranslated() → per ottenere informazioni sul path nella forma risolta

Metodi per accedere agli Header

- String getHeader(String name) → restituisce il valore di un header individuato per nome sotto forma di stringa
- Enumeration getHeaders(String name) → restituisce tutti i valori dell'header individuato da name sotto forma di enumerazione di stringhe (utile ad esempio per Accept che ammette n valori)
- Enumeration getHeaderNames() → elenco dei nomi di tutti gli header presenti nella richiesta

- int getIntHeader(name) → valore di un header convertito in intero
- long getDateHeader(name) → valore della parte Date di header, convertito in long

Autenticazione, sicurezza e cookies

- String getRemoteUser() → nome di user se la servlet ha accesso autenticato, null altrimenti
- String getAuthType() → nome dello schema di autenticazione usato per proteggere la servlet
- boolean isUserInRole(java.lang.String role) → restituisce true se l'utente è associato al ruolo specificato
- Cookie[] getCookies() → restituisce un array di oggetti cookie che il client ha inviato alla request

- I form dichiarano i campi utilizzando l'attributo name
- Quando il form viene inviato al server, nome dei campi e loro valori sono inclusi nella request:
  - agganciati alla URL come query string (GET)
  - inseriti nel body del pacchetto HTTP (POST)

- HttpRequest espone anche il metodo InputStream getInputStream();
- Consente di leggere il body della richiesta (ad esempio dati di post)

- Se non viene ridefinito, il metodo service effettua il dispatch delle richieste ai metodi doGet, doPost, … a seconda del metodo HTTP usato nella request

Deployment

- Un'applicazione Web deve essere installata e questo processo prende il nome di deployment
- Il deployment comprende:
  - La definizione del runtime environment di una Web Application
  - La mappatura delle URL sulle servlet
  - La definizione delle impostazioni di default di un'applicazione, ad es. welcome page e pagine di errore
  - La configurazione delle caratteristiche di sicurezza dell'applicazione
- Gli Archivi Web (Web Archives) sono file con estensione ".war"
  - Rappresentano la modalità con cui avviene la distribuzione/deployment delle applicazioni Web
  - Sono file jar con una struttura particolare
- La struttura di directory delle Web Application è basata sulle Servlet 2.4 specification
- web.xml è un file di configurazione (in formato XML) che descrive la struttura dell'applicazione Web
  - Contiene l'elenco delle servlet attive sul server
  - Per ogni servlet permette di definire
    - nome
    - classe Java corrispondente
    - una serie di parametri di configurazione (coppie nome-valore, valori di inizializzazione)
    - IMPORTANTE: contiene mappatura tra URL e servlet che compongono l'applicazione

web.xml | Web Application deployment descriptor

Servlet configuration

- Una servlet accede ai propri parametri di configurazione mediante l'interfaccia ServletConfig
- Ci sono 2 modi per accedere a oggetti di questo tipo:
    - Il parametro di tipo ServletConfig passato al metodo init()
    - il metodo getServletConfig()della servlet, che può essere invocato in qualunque momento
- ServletConfig espone un metodo per ottenere il valore di un parametro in base al nome: String getInitParameter(String parName)
- Esempio di parametro di configurazione:

```
<init-param>
        <param-name>parName</param-name>
        <param-value>parValue</param-value>
</init-param>
```

Servlet context

- Ogni Web application esegue in un contesto: corrispondenza 1:1 tra una Web-app e suo contesto
- L'interfaccia ServletContext è la vista della Web application (del suo contesto) da parte della servlet
- Si può ottenere un'istanza di tipo ServletContext all'interno della servlet utilizzando il metodo

Parametri di inizializzazione del contesto definiti all'interno di elementi di tipo `context-param` in `web.xml`

```
<web-app>
  <context-param>
     <param-name>feedback</param-name>
     <param-value>feedback@deis.unibo.it</param-value>
  </context-param> ...
</ web-app >
```

Sono accessibili in lettura a tutte le servlet della Web application

```
…
ServletContext ctx = getServletContext();
String feedback =
ctx.getInitParameter("feedback");
…
```

getServletContext()

- Consente di accedere ai parametri di inizializzazione e agli attributi del contesto
- Consente di accedere alle risorse statiche della Web application (es. immagini) mediante il metodo getResourceAsStream(String path)

- IMPORTANTE: servlet context viene condiviso tra tutti gli utenti, tutte le richieste e tutti i componenti server side (servlet, …) della stessa Web application
- Gli attributi di contesto sono accessibili a tutte le servlet e funzionano come variabili "globali"
- Vengono gestiti a runtime: possono essere creati, scritti e letti dalle servlet
- Possono contenere oggetti anche complessi (serializzazione/deserializzazione)

Gestione dello stato (di sessione)

- HTTP è un protocollo stateless: non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso client
- Le Applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere

- Le Applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere traccia delle informazioni di stato
    - uso dei cookie: meccanismo di basso livello
    - uso della sessione (session tracking): meccanismo di alto livello → La sessione rappresenta un'utile astrazione ed essa stessa può far ricorso a due meccanismi base di implementazione:
        - Cookie
        - URL rewriting

Cookie

- Il cookie è un'unità di informazione che Web server deposita sul Web browser lato cliente
    - Può contenere valori propri del dominio funzionale dell'applicazione (in genere informazioni associate all'utente)
    - Sono parte dell'header HTTP, trasferiti in formato testuale
    - Mandati avanti e indietro nelle richieste e nelle risposte
    - Memorizzati dal browser (client maintained state)
- Attenzione però:
    - possono essere rifiutati dal browser (tipicamente perché disabilitati)
    - sono spesso considerati un fattore di rischio
- Un cookie contiene un certo numero di informazioni, tra cui:
    - una coppia nome/valore
    - il dominio Internet dell'applicazione che ne fa uso
    - path dell'applicazione
    - una expiration date espressa in secondi (-1 indica che il cookie non sarà memorizzato su file associato)
    - un valore booleano per definirne il livello di sicurezza
- La classe Cookie modella il cookie HTTP
    - Si recuperano i cookie dalla request utilizzando il metodo getCookies()
    - Si aggiungono cookie alla response utilizzando il metodo addCookie()

Uso della sessione

- La sessione Web è un'entità gestita dal Web container
- È condivisa fra tutte le richieste dallo stesso client → consente di mantenere informazioni di stato (di sessione)
- Può contenere dati di varia natura ed è identificata in modo univoco da un session ID
- Viene usata dai componenti di una Web application per mantenere lo stato del client durante le molteplici interazioni dell'utente con la Web application

- L'accesso avviene mediante l'interfaccia HttpSession
- Per ottenere un riferimento ad un oggetto di tipo HttpSession si usa il metodo getSession() dell'interfaccia HttpServletRequest public HttpSession getSession(boolean createNew);
- Valori di createNew:
    - true: ritorna la sessione esistente o, se non esiste, ne crea una nuova

- ○ true: ritorna la sessione esistente o, se non esiste, ne crea una nuova
- ○ false: ritorna, se possibile, la sessione esistente, altrimenti ritorna null

- Uso del metodo in una servlet: HttpSession session = request.getSession(true);
- Si possono memorizzare dati specifici dell'utente negli attributi della sessione (coppie nome/valore)
- Sono simili agli attributi di contesto, ma con scope fortemente diverso!
- Consentono di memorizzare e recuperare oggetti

- String getID() → restituisce l'ID di una sessione
- boolean isNew() → dice se la sessione è nuova
- void invalidate() → permette di invalidare (distruggere) una sessione
- long getCreationTime() → dice da quanto tempo è attiva la sessione (in millisecondi)
- long getLastAccessedTime () → dà informazioni su quando è stata utilizzata l'ultima volta

- Il session ID è usato per identificare le richieste provenienti dallo stesso utente e mapparle sulla corrispondente sessione
  - ○ Una tecnica per trasmettere l'ID è quella di includerlo in un cookie (session cookie): sappiamo però che non sempre i cookie sono attivati nel browser
  - ○ Un'alternativa è rappresentata dall'inclusione del session ID nella URL: si parla di URL rewriting
    - ■ È buona prassi codificare sempre le URL generate dalle servlet usando il metodo encodeURL() di HttpServletResponse
    - ■ Il metodo encodeURL() dovrebbe essere usato per hyperlink / form

Scope differenziati (Scoped objects)

- Gli oggetti di tipo ServletContext, HttpSession, HttpServletRequest forniscono metodi per immagazzinare e ritrovare oggetti nei loro rispettivi ambiti (scope)
- Lo scope è definito dal tempo di vita (lifespan) e dall'accessibilità da parte delle servlet
- Gli oggetti scoped forniscono i sequenti metodi per immaqazzinare e ritrovare oqqetti nei rispettivi ambiti (scope):

Ambito	Interfaccia	Tempo di vita	Accessibilità
Request	`HttpServletRequest`	Fino all'invio della risposta	Servlet corrente e ogni altra pagina interrogata tramite include o forward
Session	`HttpSession`	Durata della sessione utente	Ogni richiesta dello stesso cliente
Application	`ServletContext`	Lo stesso dell'applicazione	Ogni richiesta alla stessa Web app anche da clienti diversi e per servlet diverse

  - ○ void setAttribute(String name, Object o)
  - ○ Object getAttribute(String name)
  - ○ void removeAttribute(String name)
  - ○ Enumeration getAttributeNames()

Inclusione di risorse Web

- Includere risorse Web (altre pagine, statiche o dinamiche) può essere utile quando si vogliono aggiungere contenuti creati da un'altra risorsa (ad es. un'altra servlet)
- Inclusione di risorsa statica: includiamo un'altra pagina nella nostra (ad es. banner)
- Inclusione di risorsa dinamica:
  - ○ la servlet inoltra una request ad un componente Web che la elabora e restituisce il risultato

- Il risultato viene incluso nella pagina prodotta dalla servlet

- La risorsa inclusa può lavorare con il response body (problemi comunque con l'utilizzo di cookie)
- Per includere una risorsa si ricorre a un oggetto di tipo RequestDispatcher che può essere richiesto al contesto indicando la risorsa da includere
    - Si invoca il metodo include passando come parametri request e response che vengono così condivisi con la risorsa inclusa
    - Se necessario, l'URL originale può essere salvato come un attributo di request

Forward (inoltro)

- Si usa in situazioni in cui una servlet si occupa di parte dell'elaborazione della richiesta e delega a qualcun altro la gestione della risposta
- Attenzione perché in questo caso la risposta è di competenza esclusiva della risorsa che riceve l'inoltro
- Se nella prima servlet è stato fatto un accesso a ServletOutputStream o PrintWriter si ottiene una IllegalStateException
- Si deve ottenere un oggetto di tipo RequestDispatcher da request passando come parametro il nome della risorsa
    - Si invoca quindi il metodo forward passando anche in questo caso request e response
    - Se necessario, l'URL originale può essere salvato come un attributo di request

- È anche possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina (ridirezione)

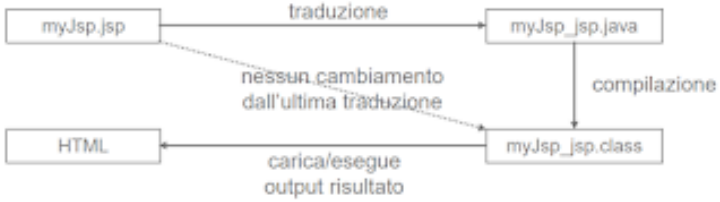
→ codici di stato HTTP da 300 a 399, in particolare 301 Moved permanently: URL non valida, server indica nuova posizione

- si fa agendo sull'oggetto response
    - metodo public void sendRedirect(String url)
    - lavorando con gli header:

```
response.setStatus(response.SC_MOVED_PERMANENTLY);
response.setHeader("Location", "http://...");
```

JAVA SERVER PAGES (JSP)

- Le JSP sono uno dei due componenti di base della tecnologia J2EE, relativamente alla parte Web:
    - template per la generazione di contenuto dinamico
    - estendono HTML con codice Java custom
- Quando viene effettuata una richiesta a una JSP:
    - parte HTML viene direttamente trascritta sullo stream di output
    - codice Java viene eseguito sul server per la generazione del contenuto HTML dinamico
    - pagina HTML così formata (parte statica + parte generata dinamicamente) viene restituita al client
- Assimilabili a linguaggio di script: in realtà vengono trasformate in servlet dal container
- Le richieste verso JSP sono gestite da una particolare servlet (in Tomcat si chiama JspServlet)

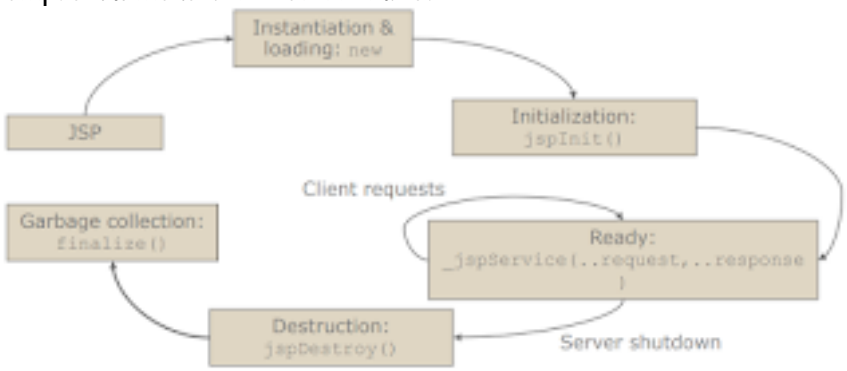


che effettua le seguenti operazioni:

- traduzione della JSP in una servlet

- compilazione della servlet risultante in una classe
- esecuzione della JSP

- I primi due passi vengono eseguiti solo quando cambia il codice della JSP
- Dal momento che JSP sono compilate in servlet, ciclo di vita delle JSP, dopo compilazione, è controllato sempre dal medesimo Web container



- Nella servlet logica per la generazione del documento HTML è implementata completamente in Java
  - Il processo di generazione delle pagine è time consuming, ripetitivo e soggetto a errori (sequenza di println())
  - L'aggiornamento delle pagine è scomodo
- JSP nascono per facilitare la progettazione grafica e l'aggiornamento delle pagine
  - Si può separare agevolmente il lavoro fra grafici e programmatori
  - Web designer possono produrre pagine senza dover conoscere i dettagli della logica server-side
  - La generazione di codice dinamico è implementata sfruttando il linguaggio Java

Servlet vs JSP:

- Le servlet forniscono agli sviluppatori delle applicazioni Web un completo controllo dell'applicazione
- JSP rendono viceversa molto semplice presentare documenti HTML o XML (o loro parti) all'utente; dominanti per la realizzazione di pagine dinamiche semplici e di uso frequente

! Come in tutti i linguaggi di script che poi generano codice, maggiori problemi di controllo della correttezza e testing

Funzionamento JSP:

- Ogni volta che arriva una request, server compone dinamicamente il contenuto della pagina
- Ogni volta che incontra un tag <%...%>
  - valuta l'espressione Java contenuta al suo interno
  - inserisce al suo posto il risultato dell'espressione
- Questo meccanismo permette di generare pagine dinamicamente

- Il Client si aspetta di ricevere tutto response header prima di response body:
  - JSP deve effettuare tutte le modifiche all'header (ad es. modifica di cookie) prima di cominciare a creare body
- Una volta che Web server comincia a restituire risposta non può più interrompere il processo, altrimenti browser mostra solo la frazione parziale che ha ricevuto:
  - se JSP ha cominciato a produrre output non si può più effettuare forward ad un'altra JSP

Tag

- Le parti variabili della pagina sono contenute all'interno di tag speciali

- Sono possibili due tipi di sintassi per questi tag:
  - Scripting-oriented tag (più diffusi): definite da delimitatori entro cui è presente lo scripting (self-contained)

    → Sono di quattro tipi:
    - <%! %> Dichiarazione → dichiara variabili e metodi, diventano parte della servlet alla traduzione
    - <%= %> Espressione → valuta espressioni Java, risultato convertito in string e inserito al posto del tag
    - <% %> Scriptlet → inserire logiche di controllo di flusso nella produzione della pagina; combinazione di tutti gli scriptlet in una determinata JSP deve definire un blocco logico completo di codice Java
    - <%@ %> Direttiva → comandi JSP valutati a tempo di compilazione, più importanti: page, include, taglib
  - XML-Oriented tag: seguono la sintassi XML

    → Sono presenti XML tag equivalenti ai delimitatori degli scripting-oriented tag:
    - <jsp:declaration>declaration</jsp:declaration>
    - <jsp:expression>expression</jsp: expression>
    - <jsp:scriptlet>java_code</jsp:scriptlet>
    - <jsp:directive.dir_type dir_attribute /

Built-in objects

Oggetto	Classe/Interfaccia
page	javax.servlet.jsp.HttpJspPage
config	javax.servlet.ServletConfig
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
pageContext	javax.servlet.jsp.PageContext
exception	Java.lang.Throwable

- Le specifiche JSP definiscono 9 oggetti built-in (o impliciti) utilizzabili senza dover creare istanze
- Rappresentano utili riferimenti ai corrispondenti oggetti Java veri e propri presenti nella tecnologia servlet
- L'oggetto page rappresenta l'istanza corrente della servlet
  - Ha come tipo l'interfaccia HTTPJspPage che discende da JSP page, la quale a sua volta estende Servlet
  - Può quindi essere utilizzato per accedere a tutti i metodi definiti nelle servlet
- L'oggetto config contiene la configurazione della servlet (parametri di inizializzazione)
  - Poco usato in pratica in quanto in generale nelle JSP sono poco usati i parametri di inizializzazione
  - Vedi anchestato a livello di applicazione in application/ServletContext
  - Metodi di config:
    - getInitParameterName(): restituisce tutti i nomi dei parametri di inizializzazione
    - getInitParameter(name): restituisce il valore del parametro passato per nome
- L'oggetto request rappresenta la richiesta alla pagina JSP
  - È il parametro request passato al metodo service() della servlet
  - Consente l'accesso a tutte le informazioni relative alla richiesta HTTP: indirizzo di provenienza, URL, headers, cookie, parametri, ecc.

Metodi di response

- `public void setHeader(String headerName, String headerValue)` imposta header

- `public void setDateHeader(String name, long millisecs)` imposta data
- `addHeader, addDateHeader, addIntHeader` aggiungono nuova occorrenza di un dato header
- `setContentType` determina content-type
- `addCookie` consente di gestire i cookie nella risposta
- `public PrintWriter getWriter`: restituisce uno stream di caratteri (un'istanza di `PrintWriter`)
- `public ServletOutputStream getOuputStream()`: restituisce uno stream di byte (un'istanza di `ServletOutputStream`)

    - Stessi metodi delle servlet
- Oggetto response è legato all'I/O della pagina JSP
    - Rappresenta la risposta che viene restituita al client
    - Consente di inserire nella risposta diverse informazioni:
        - content type ed encoding
        - eventuali header di risposta
        - URL Rewriting
        - cookie
- Oggetto out è legato all'I/O della pagina JSP, è uno stream di caratteri e rappresenta lo stream di output della pagina. Metodi:
    - isAutoFlush(): dice se output buffer è stato impostato in modalità autoFlush o meno
    - getBufferSize(): restituisce dimensioni del buffer
    - getRemaining() indica quanti byte liberi ci sono nel buffer
    - clearBuffer() ripulisce il buffer
    - flush() forza l'emissione del contenuto del buffer
    - close() fa flush e chiude stream
- Oggetto session fornisce informazioni sul contesto di esecuzione della JSP in termini di SESSIONE UTENTE
    - L'attributo session della direttiva page deve essere true affinché JSP partecipi alla sessione
        - String getID() restituisce ID di una sessione
        - boolean isNew() dice se sessione è nuova
        - void invalidate() permette di invalidare (distruggere) una sessione
        - long getCreationTime() ci dice da quanto tempo è attiva la sessione (in ms)
        - long getLastAccessedTime() ci dice quando è stata utilizzata l'ultima volta
- Oggetto application fornisce informazioni su contesto di esecuzione della JSP con scope di visibilità comune a tutti gli utenti (è ServletContext)
    - Rappresenta la Web application a cui JSP appartiene
    - Consente di interagire con l'ambiente di esecuzione:
        - fornisce la versione di JSP Container
        - garantisce l'accesso a risorse server-side
        - permette accesso ai parametri di inizializzazione relativi all'applicazione
        - consente di gestire gli attributi di un'applicazione
- Oggetto pageContext fornisce informazioni sul contesto di esecuzione della pagina JSP
    - Rappresenta l'insieme degli oggetti built-in di una JSP
    - Consente accesso a tutti gli oggetti impliciti e ai loro attributi
    - Consente trasferimento del controllo ad altre pagine
    - Nota: poco usato in caso di scripting, più utile per costruire custom tag
- Oggetto exception connesso alla gestione degli errori
    - Rappresenta l'eccezione che non viene gestita da nessun blocco catch
    - Non è automaticamente disponibile in tutte le pagine ma solo nelle Error Page (quelle dichiarate con l'attributo errorPage impostato a true)

dichiarate con l'attributo errorPage impostato a true)

Azioni

- Comandi JSP per l'interazione con altre pagine JSP, servlet, o componenti JavaBean; espresse usando sintassi XML
- Previsti 6 tipi di azioni definite dai seguenti tag:
    - useBean: istanzia JavaBean e gli associa un identificativo
    - getProperty: ritorna property indicata come oggetto
    - setProperty: imposta valore della property indicata per nome
    - include: include nella JSP contenuto generato dinamicamente da un'altra pagina locale
        - Sintassi: <jsp:include page="localURL" flush="true" />
        - Trasferisce temporaneamente controllo ad un'altra pagina
        - L'attributo page definisce l'URL della pagina da includere
        - L'attributo flush stabilisce se sul buffer della pagina corrente debba essere eseguito flush prima di effettuare l'inclusione
        - Gli oggetti session e request per pagina da includere sono gli stessi della pagina chiamante, ma viene istanziato un nuovo contesto di pagina
    - forward: cede controllo ad un'altra JSP o servlet
        - Sintassi: <jsp:forward page="localURL" />
        - L'attributo page definisce l'URL della pagina a cui trasferire il controllo
        - La request viene completamente trasferita in modo trasparente per il client
        - Oggetti request, response e session della pagina d'arrivo sono gli stessi della pagina chiamante, ma viene istanziato un nuovo oggetto pageContext
        - Attenzione: forward è possibile soltanto se non è stato emesso alcun output
    - plugin: genera contenuto per scaricare plug-in Java se necessario
- È possibile aggiungere parametri all'oggetto request della pagina inclusa utilizzando il tag <jsp:param>

JavaBeans

- JavaBeans è il modello di "base" per componenti Java, il più semplice
- Un JavaBean, o semplicemente bean, non è altro che una classe Java dotata di alcune caratteristiche particolari:
    - Classe public
    - Ha un costruttore public di default (senza argomenti)
    - Espone proprietà, sotto forma di coppie di metodi di accesso (accessors) costruiti secondo le regole che abbiamo appena esposto (get... set...)
    - Espone eventi con metodi di registrazione che seguono regole precise
- proprietà sono elementi dello stato del componente che vengono esposti in modo protetto
    - In alcuni linguaggi (ad esempio C#) esiste sintassi specifica per definire le proprietà
- proprietà prop è definita da due metodi getProp() e setProp()
    - Il tipo del parametro di setProp() e del valore di ritorno di getProp() devono essere uguali e rappresentano il tipo della proprietà (può essere un tipo primitivo o una qualunque classe Java)
    - Per esempio void setLength(int Value) e int getLength() identificano proprietà length di tipo int
- Se definiamo solo il metodo get avremo una proprietà in sola lettura (read-only)
- Le proprietà di tipo boolean seguono una regola leggermente diversa: metodo di lettura ha la forma isProp() anziché getProp()
- possibilità di definire proprietà indicizzate per rappresentare collezioni di valori (pseudoarray). In questo caso sia get che set prevedono un parametro che ha la funzione di indice
    - es. String getItem(int index) e setItem(int Index, String value) definiscono proprietà

- es. String getItem(int index) e setItem(int index, String value) definiscono proprietà indicizzata String item[]

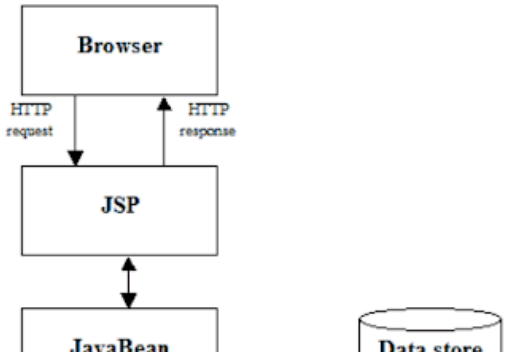
- I componenti vivono all'interno di contenitori (component container) che gestiscono:
  - tempo di vita dei singoli componenti
  - collegamenti tra componenti e resto del sistema
- I contenitori non conoscono a priori i componenti che devono gestire e quindi interagiscono con loro mediante meccanismi di tipo dinamico (spesso reflection)
- Un contenitore per JavaBean prende il nome di bean container
- Un bean container è in grado di interfacciarsi con i bean utilizzando Java Reflection che fornisce strumenti di introspezione e di dispatching
- L'obbligo del costruttore di default ha proprio lo scopo di consentire creazione dinamica delle istanze
- JSP prevedono una serie di tag per agganciare un bean e utilizzare le sue proprietà all'interno della pagina. Tre tipi:
  - Tag per creare un riferimento al bean (creazione di un'istanza)
  - Tag per impostare il valore delle proprietà del bean
  - Tag per leggere il valore delle proprietà del bean e inserirlo nel flusso della pagina

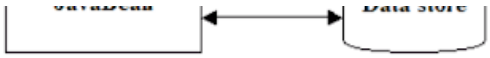
Tempo di vita dei bean

- Per default ogni volta che una pagina JSP viene richiesta e processata viene creata un'istanza del bean (scope di default = page)
- Con l'attributo scope è possibile estendere la vita del bean oltre la singola richiesta

- useBean
  - <jsp:useBean id="beanName" class="class" scope="page|request|session|application"/>
  - Inizializza e crea il riferimento al bean
  - Gli attributi principali sono id, class e scope
  - id è il nome con cui l'istanza del bean verrà indicata nel resto della pagina
  - class è classe Java che definisce il bean
  - scope definisce ambito di accessibilità e tempo di vita dell'oggetto (default = page)
- getProperty
  - <jsp:getProperty name="beanName" property="propName"/>
  - Consente l'accesso alle proprietà del bean
  - Produce come output il valore della proprietà del bean
  - Il tag non ha mai body e ha solo 2 attributi:
    - name: nome del bean a cui si fa riferimento
    - property: nome della proprietà di cui si vuole leggere il valore
- setProperty
  - <jsp:setProperty name="beanName" property="propName" value="propValue"/>
  - Consente di modificare il valore delle proprietà del bean

- I taq per JavaBean non supportano proprietà indicizzate





- Però un bean è un normale oggetto Java: è quindi possibile accedere a variabili e metodi
- L'architettura J2EE a due livelli costituita da
    - JSP per il livello di presentazione
    - JavaBean per il livello di business logic viene denominata Model 1

Domande:

- Possiamo avere una applicazione Web che include più servlet e più JSP?
- Posso fare «forward» e «include» da JSP a servlet e da servlet a JSP?
- Posso vedere un attributo di sessione definito in una servlet all'interno di una JSP?
- Posso vedere un attributo di applicazione definito in una servlet all'interno di una JSP?
- Posso vedere un Javabean definito in una JSP all'interno di una servlet? Con quali scope?

Appendice: Custom tag e tag libraries

- JSP permettono di definire tag personalizzati (custom tag) che estendono quelli predefiniti
- Una taglib è una collezione di questi tag non standard, realizzata mediante una classe Java
- Per utilizzarla si usa la direttiva taglib con la sintassi: <%@ taglib uri="tagLibraryURI" prefix="tagPre"%>
    - L'attributo uri fa riferimento ad un file xml, con estensione tld (tag library descriptor), che contiene informazioni sulle classi che implementano i tag
    - L'attributo prefix indica il prefisso da utilizzare nei tag che fanno riferimento alla tag library (tag library è un namespace)
- Per definire una tag library occorrono due elementi:
    - File TLD (Tag Lib Definition) che specifica i singoli Tag e a quale "classe" corrispondono
    - Le classi che effettivamente gestiscono i tag
- File TLD è un file XML che specifica:
    - i tag che fanno parte della libreria
    - i loro eventuali attributi
    - "body" del tag (se esiste)
    - classe Java che gestisce il tag
- Dovremo quindi sviluppare le classi che implementano il comportamento dei tag
- Una singola libreria può contenere centinaia di tag o uno solo

- Innanzitutto inseriamo nella JSP la direttiva che include la libreria di tag:

<%@ taglib uri="hellolib.tld" prefix="htl" %>

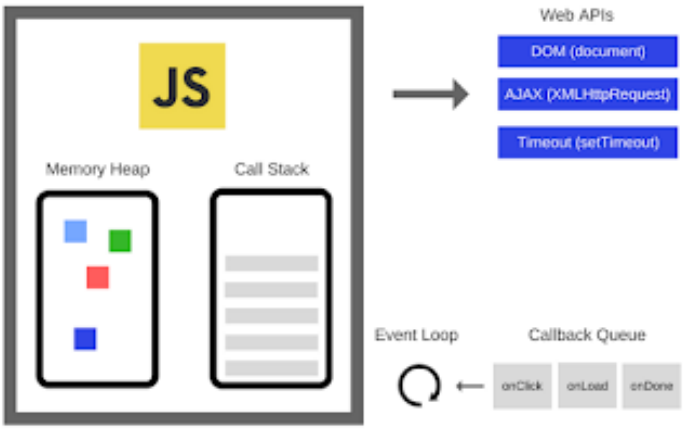
- Il prefisso definisce un namespace e quindi elimina le eventuali omonimie causate dall'inclusione di più librerie. Possiamo quindi usare il tag con la sintassi:

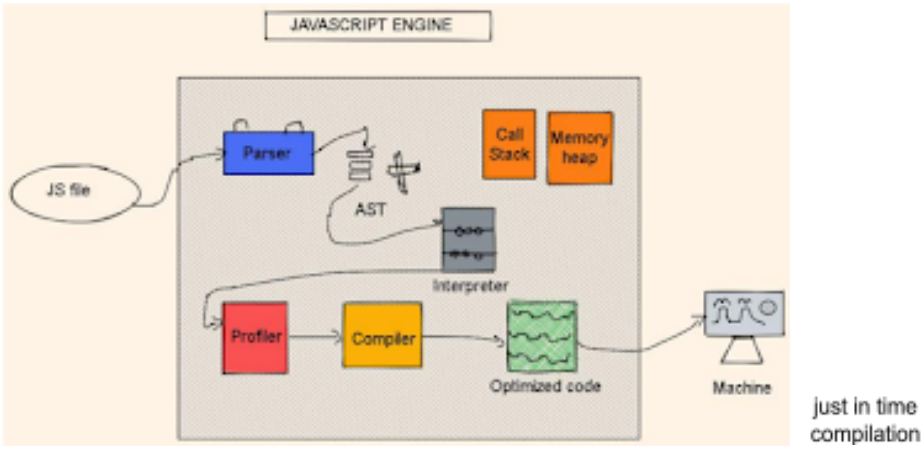
<htl:helloWorld who="Mario">

- Per implementare tag dobbiamo scrivere una apposita classe Java che estende TagSupport, la classe base per i tag "semplici", per quelli complessi sono disponibili altre classi base
- La classe deve implementare
    - i metodi doStartTag()e doEndTag()
    - Una coppia di metodi di accesso (setAttrName() e getAttrName()per ogni attributo
    - doStartTag() utilizza l'oggetto out restituito da PageContext per scrivere nell'output della pagina e se tag non ha nessun "body" deve ritornare come valore la costante SKIP_BODY
    - doEndTag()restituisce usualmente la costante EVAL_PAGE che indica che, dopo il tag, prosegue la normale elaborazione della pagina

JAVASCRIPT

- linguaggio di scripting sviluppato per dare interattività LATO CLIENTE alle pagine HTML → Pagine Web attive
- Può essere inserito direttamente nelle pagine Web
- Java vs JavaScript:
    - ○ JavaScript è interpretato e non compilato (interprete contenuto nel browser)
    - ○ JavaScript è object-based ma non class-based: esiste il concetto di oggetto ma non esiste il concetto di classe
    - ○ JavaScript è debolmente tipizzato (weakly typed): non è necessario definire il tipo di una variabile
        - ■ Attenzione però: questo non vuol dire che i dati non abbiano un tipo (sono le variabili a non averlo in modo statico)





- Nasce per dare dinamicità alle pagine Web (pagine attive), consente quindi di:
    - ○ Accedere e modificare elementi della pagina HTML
    - ○ Reagire ad eventi generati dall'interazione con l'utente
    - ○ Validare i dati inseriti dall'utente
    - ○ Interagire con il browser: e.g., determinare il browser utilizzato, la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc.
- La sintassi di JavaScript è modellata su quella del C con alcune varianti significative, in particolare:
    - ○ È un linguaggio case-sensitive
    - ○ Le istruzioni sono terminate da ';' ma il terminatore può essere omesso se si va a capo
    - ○ Sono ammessi sia commenti multilinea (delimitati da /* e */) che mono-linea (iniziano con //)
    - ○ Gli identificatori possono contenere lettere, cifre e i caratteri '_' e '$' ma non possono

- Gli identificatori possono contenere lettere, cifre e i caratteri "_" e "$" ma non possono iniziare con una cifra

Variabili

- Le variabili vengono dichiarate usando la parola chiave var: var nomevariabile;
- Non hanno un tipo → possono contenere valori di qualunque tipo
- È prevista la possibilità di inizializzare una variabile contestualmente alla dichiarazione var f = 15.8
- Possono essere dichiarate in linea: for (var i = 1, i<10, i++)
- Esiste scope globale e locale (dentro una funzione) ma, a differenza di Java, non esiste lo scope di blocco (eccezione: let)
- Ad ogni variabile può essere assegnato il valore null che rappresenta l'assenza di un valore
→ concetto diverso da zero (0) o stringa vuota ("")
- Una variabile non inizializzata ha invece un valore indefinito undefined

! I due concetti si assomigliano ma non sono uguali

Tipi

- Javascript prevede pochi tipi primitivi: numeri, booleani e stringhe
- Numeri (number):
  - Sono rappresentati in formato floating point a 8 byte, non c'è distinzione fra interi e reali
  - Esiste il valore NaN (not a number) per le operazioni non ammesse (ad esempio, radice di un numero negativo)
  - Esiste il valore infinite (ad esempio, per la divisione per zero)
- Booleani (boolean):
  - ammettono i valori true e false
- Come abbiamo detto, alle variabili non viene attribuito un tipo: lo assumono dinamicamente in base al dato a cui vengono agganciate
- I dati hanno un tipo; per ogni tipo esiste una sintassi per esprimere le costanti
→ per i numeri, ad esempio, le costanti hanno la forma usuale: 1.0, 3.5 o in altre basi
→ per i booleani sono gli usuali valori true e false

Oggetti e array

- Gli oggetti sono tipi composti che contengono un certo numero di proprietà (attributi)
- Ogni proprietà ha un nome e un valore
- Si accede alle proprietà con l'operatore '.' (punto)
- Le proprietà non sono definite a priori, possono essere aggiunte dinamicamente
- Gli oggetti vengono creati usando l'operatore new: var o = new Object()

! Attenzione: Object() è un costruttore e non una classe. Le classi non esistono, i due concetti non si sovrappongono come avviene in Java

- Un oggetto appena creato è completamente vuoto: non ha né proprietà né metodi
- Possiamo costruirlo dinamicamente: appena assegniamo un valore ad una proprietà la proprietà comincia ad esistere
- Le costanti oggetto (object literal) sono racchiuse fra parentesi graffe { } e contengono un elenco di attributi nella forma: nome:valore

var nomeoggetto = {prop1:val1, prop2:val2, ...}

- Usando le costanti oggetto creiamo un oggetto e le proprietà (valorizzate) nello stesso momento
- Gli array sono tipi composti i cui elementi sono accessibili mediante un indice numerico
  - l'indice parte da zero
  - non hanno una dimensione prefissata (simili agli ArrayList di Java)
  - espongono attributi e metodi

- Vengono istanziati con new Array([dimensione])
- Si possono creare e inizializzare usando delle costanti array (array literal) delimitate da [ ]: var name = [val,val2,…,valn]

⟶ Es. var a = [1,2,3];

- Possono contenere elementi di tipo eterogeneo: var b = [1,true,"ciao",{x:1,y:2}];
- Gli oggetti in realtà sono array associativi: strutture composite i cui elementi sono accessibili mediante un indice di tipo stringa (nome) anziché attraverso un indice numerico

⟶ Si può quindi utilizzare anche una sintassi analoga a quella degli array

```
var o = new Object();
o.x = 7;
o.y = 8;
o.tot = o.x + o.y;
alert(o.tot);
```

```
var o = new Object();
o["x"] = 7;
o.y = 8;
o["tot"] = o.x + o["y"];
alert(o.tot);
```

Stringhe

- Non è facile capire esattamente cosa sono le stringhe in JavaScript
- Potremmo dire che mentre in Java sono oggetti che sembrano dati di tipo primitivo in JavaScript sono dati di tipo primitivo che sembrano oggetti
- Sono sequenze arbitrarie di caratteri in formato UNICODE a 16 bit e sono immutabili come in Java
- Esiste la possibilità di definire costanti stringa (string literal) delimitate da apici singoli ('ciao') o doppi ("ciao")
- È possibile la concatenazione con l'operatore + o la comparazione con gli operatori <, >, >=, <= e !=
- Possiamo però invocare metodi su una stringa o accedere ai suoi attributi: var t = s.charAt(1);
- Non sono però oggetti e la possibilità di trattarli come tali nasce da due caratteristiche:
    - Esiste un tipo wrapper String che è un oggetto
    - JavaScript fa il boxing in automatico come C#
        - quando una variabile di tipo valore necessita di essere convertita in tipo riferimento, un oggetto box è allocato per mantenere tale valore

Espressioni regolari

- JavaScript ha un supporto per le espressioni regolari (regular expressions) che sono un tipo di dato nativo del linguaggio
- Esistono le costanti di tipo espressione regolare (regexp literal) con la sintassi / expression /
- Una espressione regolare può essere creata anche mediante il costruttore RegExp()

- Si può tentare di interpretare il sistema dei tipi di JavaScript usando una logica simile a quella di C#
- Si può quindi distinguere tra tipi valore e tipi riferimento
    - Numeri e booleani sono tipi valore
    - Array e Oggetti sono tipi riferimento
    - Per le stringhe abbiamo ancora una situazione incerta: pur essendo un tipo primitivo si comportano come un tipo riferimento

Funzioni

- Una funzione è un frammento di codice JavaScript che viene definito una volta e usato in più punti
- Ammette parametri che sono privi di tipo
- Restituisce un valore il cui tipo non viene definito
- La mancanza di tipo è coerente con la scelta fatta per le variabili

- Le funzioni possono essere definite utilizzando la parola chiave function
- Una funzione può essere assegnata ad una variabile
- Esistono costanti funzione (function literal) che permettono di definire una funzione e poi di assegnarla ad una variabile con una sintassi decisamente inusuale:

```
var sum = function(x,y) { return x+y; }
```

- Una funzione può essere anche creata usando un costruttore denominato Function (le funzioni sono quindi equivalenti in qualche modo agli oggetti)

```
var sum = new Function("x","y","return x+y;");
```

Metodi

- Quando una funzione viene assegnata ad una proprietà di un oggetto viene chiamata metodo dell'oggetto
- La cosa è possibile perché, come abbiamo visto, una funzione può essere assegnata ad una variabile
- In questo caso all'interno della funzione si può utilizzare la parola chiave this per accedere all'oggetto di cui la funzione è una proprietà

Costruttori

- Un costruttore è una funzione che ha come scopo quello di costruire un oggetto
- Se viene invocato con new riceve l'oggetto appena creato e può aggiungere proprietà e metodi
- L'oggetto da costruire è accessibile con this
- Si può dire che il costruttore definisce il tipo di un oggetto

- JavaScript ammette l'esistenza di proprietà e metodi statici con lo stesso significato di Java
- Non esistendo le classi sono associati al costruttore
- Anche in Javascript esiste l'oggetto Math che definisce solo metodi statici corrispondenti alle varie funzioni matematiche

Istruzioni

- Espressioni (uguali a Java): assegnamenti, invocazioni di funzioni e metodi, ecc.
- Istruzioni composte: blocchi di istruzioni delimitate da parentesi graffe (uguali a Java)
- Istruzione vuota: punto e virgola senza niente prima
- Istruzioni etichettate: normali istruzioni con un'etichetta davanti (sintassi: label: statement)
- Strutture di controllo: if, for, while, ecc.
- Definizioni e dichiarazioni: var, function
- Istruzioni speciali: break, continue, return

Oggetto globale

- In JavaScript esiste un oggetto globale implicito
- Tutte le variabili e le funzioni definite in una pagina appartengono all'oggetto globale
- Possono essere utilizzate senza indicare questo oggetto
- Questo oggetto espone anche alcune funzioni predefinite:
    - eval(expr)valuta la stringa expr (che contiene un'espressione Javascript)
    - isFinite(number) dice se il numero è finito
    - isNaN(testValue) dice se il valore è NaN
    - parseInt(str [,radix]) converte la stringa str in un intero (in base radix - opzionale)
    - parseFloat(str): converte la stringa str in un numero

Includere JavaScript in una pagina HTML

- HTML prevede un apposito tag per inserire script; la sua sintassi è <script> <!-- script-text //-->

</script>

- Il commento HTML (<!-- //-->) che racchiude il testo dello script serve per gestire la compatibilità con i browser che non gestiscono JavaScript

→ In questi casi il contenuto del tag viene ignorato

- La sintassi completa prevede anche la definizione del tipo di script definito (Javascript è il default per gran parte dei browser); si può fare in tre modi:
  - <script language="Javascript"> (deprecato) oppure
  - <script type="text/javascript"> (rif. HTML 4) (deprecato)
  - <script type="application/javascript"> (rif. HTML 5)
- Nell'uso del tag <script> abbiamo due possibilità:
  - Script esterno: il tag contiene il riferimento ad un file con estensione .js che contiene lo script: <SCRIPT language="Javascript" src="nomefile.js"></SCRIPT>
  - Script interno: lo script è contenuto direttamente nel tag:
    <script type="text/javascript"> alert("Hello World!"); </script>
- Se lo script è interno può essere inserito sia nell'intestazione che nel body
- Una pagina HTML viene eseguita in ordine sequenziale, dall'alto verso il basso, per cui:
  - gli script di intestazione vengono caricati prima di tutti gli altri
  - quelli nel body vengono eseguiti secondo l'ordine di caricamento
- Una variabile o qualsiasi altro elemento Javascript può essere richiamato solo se caricato in memoria:
  - ciò che si trova nell'header è visibile a tutti gli script del body
  - quello che si trova nel body è visibile solo agli script che lo seguono
- Attenzione: questi script vengono eseguiti solo una volta durante il caricamento della pagina e quindi non si ha interattività con l'utente
- HTML prevede un tag (<noscript>) da inserire in testata per gestire contenuti alternativi in caso Javascript non sia gestito dal browser utilizzato

- Per interagire con la pagina HTML , Javascript utilizza una gerarchia di oggetti predefiniti denominati Browser Objects e DOM Objects

Modello ad eventi

- Per avere una reale interattività bisogna utilizzare il meccanismo degli eventi
- JavaScript consente di associare script agli eventi causati dall'interazione dell'utente con la pagina HTML
- L'associazione avviene mediante attributi collegati agli elementi della pagina HTML
- Gli script prendono il nome di gestori di eventi (event handlers)
- Nelle risposte agli eventi si può intervenire sul DOM modificando dinamicamente la struttura della pagina (DHTML)

DHTML = JavaScript + DOM + CSS

- È un modello di tipo reattivo simile a quello di Swing o delle applicazioni Windows sviluppate con .NET
- Sintassi: <tag eventHandler="JavaScript Code">

Document

- Il punto di partenza per accedere al Documento Object Model (DOM) della pagina è l'oggetto document
- document espone 4 collezioni di oggetti che rappresentano gli elementi di primo livello:
  - anchors[]
  - forms[]
  - images[]
  - links[]

- L'accesso agli elementi delle collezioni può avvenire per indice (ordine di definizione nella pagina) o per nome (attributo name dell'elemento): document.links[0] document.links["nomelink"]
- In base all'equivalenza tra array associativi e oggetti la seconda forma può essere scritta anche come document.nomelink
- Metodi:
    - getElementById(): restituisce riferimento al primo oggetto della pagina avente l'id specificato come argomento
    - write(): scrive un pezzo di testo nel documento
    - writeln(): come write() ma aqqiunqe un a capo

Validazione di un form

- Uno degli utilizzi più frequenti di JavaScript è nell'ambito della validazione dei campi di un form
    - Riduce il carico delle applicazioni server side filtrando l'input
    - Riduce il ritardo in caso di errori di inserimento dell'utente
    - Semplifica le applicazioni server side
    - Consente di introdurre dinamicità all'interfaccia Web
- Generalmente si valida un form in due momenti:
    - Durante l'inserimento utilizzando l'evento `onChange()` sui vari controlli
    - Al momento del submit utilizzando l'evento `onClick()` del bottone di submit o l'evento `onSubmit()` del form

- Proprietà:
    - bgcolor: colore di sfondo
    - fgcolor: colore di primo piano
    - lastModified: data e ora di ultima modifica
    - cookie: tutti i cookie associati al document, rappresentati da una stringa di coppie nome-valore
    - title: titolo del documento
    - URL: url del documento

JQuery

- Sinteticamente, JQuery è una libreria JavaScript (sviluppata da terzi) pensata appositamente per semplificare la vita del programmatore Web
- Nel dettaglio, JQuery semplifica e velocizza
    - l'attraversamento del DOM di una pagine HTML,
    - la sua animazione,
    - la gestione di eventi,
    - le interazioni Ajax

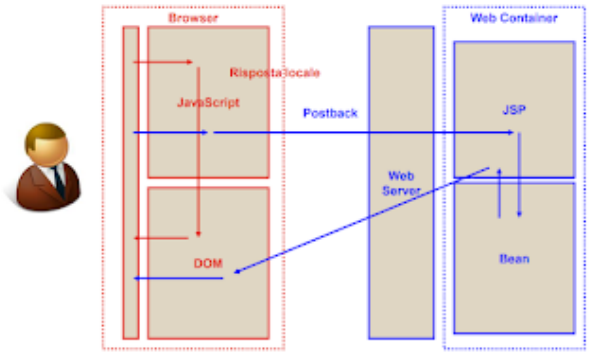
mediante una "easy-to-use" API che funziona per una moltitudine di Web browser
- Mediante una giusta combinazione di versatilità e estensibilità, JQuery ha cambiato il modo di scrivere codice JavaScript
- Prima di JQuery, gli sviluppatori tendevano a creare il proprio "framework JavaScript"; ciò permetteva loro di lavorare su specifici bug senza perdere tempo nel debugging di feature comuni
- Ciò ha portato alla realizzazione da parte di gruppi di sviluppatori di librerie open source e gratuite
- Con JQuery, lo sviluppatore usa API JavaScript preconfezionate «pronte all'uso»

AJAX

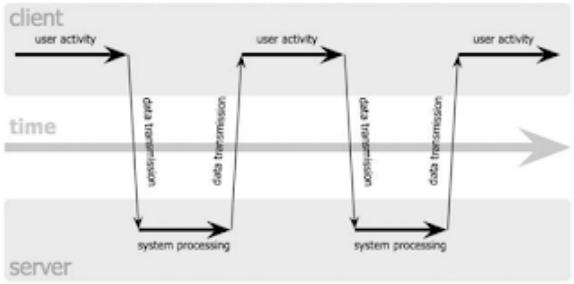
- L'utilizzo di DHTML (JavaScript/Eventi + DOM + CSS) delinea un nuovo modello per applicazioni Web

=> Modello a eventi simile a quello delle applicazioni tradizionali

Modello a eventi a due livelli



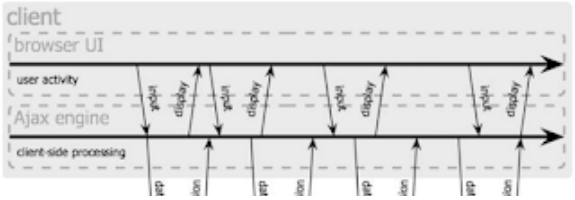
- A livello concettuale abbiamo però due livelli di eventi:
  - Eventi locali che portano ad una modifica diretta DOM da parte di Javascript e quindi a cambiamento locale della pagina
  - Eventi remoti ottenuti tramite ricaricamento della pagina che viene modificata lato server in base ai parametri passati in GET o POST
- Il ricaricamento di pagina per rispondere a interazione con l'utente prende il nome di postback
- Quando lavoriamo con applicazioni desktop siamo abituati a un elevato livello di interattività:
  - applicazioni reagiscono in modo rapido e intuitivo ai comandi Applicazioni Web tradizionali esponqono invece un modello di interazione rigido

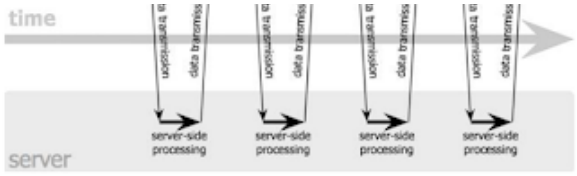


  - Modello "Click, wait, and refresh": è necessario refresh della pagina da parte del server per la gestione di qualunque evento (sottomissione di dati tramite form, visita di link per ottenere informazioni di interesse, ...)

    → È comunque modello sincrono: l'utente effettua una richiesta e deve attendere la risposta da parte del server

- AJAX è nato per superare queste limitazioni
- AJAX non è un acronimo ma spesso viene interpretato come Asynchronous Javascript And Xml
- AJAX non è una nuova tecnologia per sé ma è basato su tecnologie standard e combinate insieme per realizzare un modello di interazione più ricco: JavaScript, DOM, XML, HTML, CSS
- AJAX punta a supportare applicazioni user friendly con elevata interattività (si usa spesso il termine RIA - Rich Interface Application)
- L'idea alla base di AJAX è quella di consentire agli script JavaScript di interagire direttamente con il server
- Elemento centrale è utilizzo dell'oqqetto JavaScript XMLHttpRequest





- ○ Consente di ottenere dati dal server senza necessità di ricaricare l'intera pagina
- ○ Realizza comunicazione asincrona tra client e server: il client non interrompe interazione con utente anche quando è in attesa di risposte dal server

Tipica sequenza Ajax:

- Si verifica un evento determinato dall'interazione fra utente e pagina Web
- L'evento comporta l'esecuzione di una funzione JavaScript in cui:
  - ○ Si istanzia un oggetto di classe XMLHttpRequest
  - ○ Si configura XMLHttpRequest: si associa una funzione di callback, si effettua configurazione, ...
  - ○ Si effettua chiamata asincrona al server
- Il server elabora la richiesta e risponde al client
- Il browser invoca la funzione di callback che:
  - ○ elabora il risultato
  - ○ aggiorna il DOM della pagina per mostrare i risultati dell'elaborazione

HTMLHttpRequest

- È l'oggetto XMLHttpRequest che effettua richiesta di una risorsa via HTTP a server Web
  - ○ NON sostituisce URI della propria richiesta all'URI corrente; NON provoca cambio di pagina
  - ○ Può inviare info (parametri) sotto forma di variabili (come form)
  - ○ Più risorse richieste concorrentemente? È sufficiente un unico oggetto XMLHttpRequest?
- Può effettuare sia richieste GET che POST
- Le richieste possono essere di tipo
  - ○ Sincrono: blocca flusso di esecuzione del codice Javascript (ci interessa poco)
  - ○ Asincrono: NON interrompe il flusso di esecuzione del codice Javascript né le operazioni dell'utente sulla pagina
  - → quindi thread dedicato
- I browser recenti supportano XMLHttpRequest come oggetto nativo
- In questo caso (oggi il più comune) le cose sono molto semplici: var xhr = new XMLHttpRequest();
- La gestione della compatibilità con browser «molto» vecchi complica un po' le cose (necessità di accesso universale da sistemi legacy long-lived);
- Attenzione: per motivi di sicurezza XmlHttpRequest può essere utilizzata solo verso dominio da cui proviene la risorsa che la utilizza
- La lista dei metodi disponibili è diversa da browser a browser In genere si usano solo quelli presenti in Safari (sottoinsieme più limitato, ma comune a tutti i browser che supportano AJAX):
  - ○ open() → inizializzare la richiesta da formulare al server
  - ○ setRequestHeader() → consente di impostare gli header HTTP della richiesta da inviare (invocata più volte, una per ogni header da impostare)
  - ○ send() → consente di inviare la richiesta al server
  - ○ getResponseHeader()
  - ○ getAllResponseHeaders()
  - ○ abort()
- Stato e risultati della richiesta vengono memorizzati dall'interprete Javascript all'interno

dell'oggetto XmlHttpRequest durante la sua esecuzione

- Le proprietà comunemente supportate dai vari browser sono:
  - readyState → proprietà in sola lettura di tipo intero che consente di leggere in ogni momento lo stato della richiesta (0: uninitialized; 1: open; 2: sent; 3: receiving; 4: loaded)
  - onreadystatechange → funzione di callback che viene richiamata in modo asincrono ad ogni cambio di stato della proprietà ReadyState
    - Attenzione: per evitare comportamenti imprevedibili l'assegnamento va fatto prima del send()
  - status → contiene un valore intero corrispondente al codice HTTP dell'esito della richiesta
  - statusText → contiene invece una descrizione testuale del codice HTTP restituito dal server
  - responseText → stringa che contiene il body della risposta HTTP
  - responseXML → body della risposta convertito in documento XML (se possibile)
- I metodi getResponseHeader(), getAllResponseHeaders() consentono di leggere gli header HTTP che descrivono la risposta del server
  - sono utilizzabili solo nella funzione di callback
  - possono essere invocati sicuramente in modo safe solo a richiesta conclusa (readystate==4)
  - in alcuni browser possono essere invocati anche in fase di ricezione della risposta (readystate==3)
- La funzione di callback:
  - Viene invocata ad ogni variazione di readystate
  - Usa readystate per leggere lo stato di avanzamento della richiesta
  - Usa status per verificare l'esito della richiesta
  - Ha accesso agli header di risposta rilasciati dal server con getAllResponseHeaders() e getResponseHeader()
  - Se readystate==4 può leggere il contenuto della risposta con responseText e responseXML

Vantaggi e svantaggi

- Si guadagna in espressività, ma si perde la linearità dell'interazione
- Mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e indipendenti
- Il tempo di attesa passa in secondo piano o non è avvertito affatto
- Possibili criticità sia per l'utente che per lo sviluppatore
  - percezione che non stia accadendo nulla (sito che non risponde)
  - problemi nel gestire un modello di elaborazione che ha bisogno di aspettare i risultati delle richieste precedenti
- Le richieste AJAX permettono all'utente di continuare a interagire con la pagina
- Ma non necessariamente lo informano di che cosa stia succedendo e possono durare troppo!

=> L'effetto è un possibile disorientamento dell'utente

- Di conseguenza, di solito si agisce su due fronti per limitare i comportamenti impropri a livello utente:
  - Rendere visibile in qualche modo l'andamento della chiamata (barre di scorrimento, info utente, ...)
  - Interrompere richieste che non terminano in tempo utile per sovraccarichi server o problemi di rete (timeout)

problemi di rete (timeout)

→ abort() consente l'interruzione delle operazioni di invio o ricezione, non ha bisogno di parametri

- ○ termina immediatamente la trasmissione dati
- ○ Attenzione: non ha senso invocarlo dentro la funzione di callback
- ○ Se readyState non cambia, metodo non viene richiamato; readyState non cambia quando risposta si fa attendere
- ○ Si crea un'altra funzione da far richiamare in modo asincrono al sistema mediante il metodo setTimeOut()
- ○ Al suo interno si valuta se continuare l'attesa o abortire l'operazione

- È accresciuta la complessità delle Web Application
- La logica di presentazione è ripartita fra client-side e server-side
- Applicazioni AJAX pongono problemi di debug, test e mantenimento
  - ○ Il test di codice JavaScript è complesso
  - ○ Il codice JavaScript ha problemi di modularità
  - ○ I toolkit AJAX sono molteplici e solo recentemente hanno raggiunto una discreta maturità
  - ○ Mancanza di standardizzazione di XMLHttpRequest e assenza di supporto nei vecchi browser

Gestire la risposta

- Spesso i dati scambiati tra client e server sono codificati in XML
- È possibile elaborare i documenti XML ricevuti utilizzando API W3C DOM
- Il modo con cui operiamo su dati in formato XML è analogo a quello che abbiamo visto per ambienti Java
  - ○ Usiamo un parser e accediamo agli elementi di nostro interesse
  - ○ Per visualizzare i contenuti ricevuti modifichiamo il DOM della pagina HTML
- utilizzo di XML come formato di scambio fra client e server porta a generazione e utilizzo di quantità di byte piuttosto elevate e non ottimizzate
- Non semplicissimo da leggere e da manutenere
- Oneroso in termini di risorse di elaborazione (non dimentichiamo che JavaScript è interpretato)

→ Soluzione: JSON

JSON

- JSON è l'acronimo di JavaScript Object Notation
- Formato per lo scambio di dati, considerato molto più comodo di XML
- Leggero in termini di quantità di dati scambiati
- Molto semplice ed efficiente da elaborare da parte del supporto runtime al linguaggio di programmazione (in particolare per JavaScript)
- Ragionevolmente semplice da leggere per operatore umano
- È largamente supportato dai maggiori linguaggi di programmazione
- Si basa sulla notazione usata per le costanti oggetto (object literal) e le costanti array (array literal) in JavaScript
- La sintassi JSON si basa su quella delle costanti oggetto e array di JavaScript
- Un "oggetto JSON" altro non è che una stringa equivalente a una costante oggetto di JavaScript
- JavaScript mette a disposizione la funzione eval() che invoca l'interprete per la traduzione della stringa passata come parametro
  - ○ La sintassi di JSON è un sottoinsieme di JavaScript: con eval possiamo trasformare una stringa JSON in un oggetto
  - ○ La sintassi della stringa passata a eval deve essere '(espressione)': dobbiamo quindi racchiudere la stringa JSON fra parentesi tonde

- Uso di eval() presenta rischi: stringa passata come parametro potrebbe contenere codice malevolo
→ Di solito si preferisce utilizzare parser appositi che traducono solo oggetti JSON e non espressioni JavaScript di qualunque tipo

Utilizzo Ajax e JSON

- Sul lato client:
  - Si crea un oggetto JavaScript e si riempiono le sue proprietà con le informazioni necessarie
  - Si usa JSON.stringify() per convertire l'oggetto in stringa JSON
  - Si usa la funzione encodeURIComponent() per convertire la stringa in un formato utilizzabile in una richiesta HTTP (vedi esercitazione su AJAX)
  - Si manda la stringa al server mediante XMLHttpRequest (stringa viene passata come variabile con GET o POST)
- Sul lato server:
  - Si decodifica la stringa JSON e la si trasforma in oggetto Java utilizzando un apposito parser (si trova sempre su www.json.org; ne parleremo anche nella esercitazione di lab dedicata)
  - Si elabora l'oggetto
  - Si crea un nuovo oggetto Java che contiene dati della risposta
  - Si trasforma l'oggetto Java in stringa JSON usando il parser suddetto
  - Si trasmette la stringa JSON al client nel corpo della risposta HTTP: response.out.write(strJSON);
- Sul lato client, all'atto della ricezione:
  - Si converte la stringa JSON in un oggetto Javascript usando JSON.parse()
  - Si usa liberamente l'oggetto per gli scopi desiderati

- Gson è una libreria java per il parsing/deparsing di oggetti JSON

In versioni recenti di Gson (a partire da 2.8):

Inizializzazione dell'oggetto Gson:

```
Gson g = new Gson();
```

Serializzazione di un oggetto:

```
Person santa = new Person("Santa", "Claus", 1000);
g.toJson(santa);
```

Deserializzazione di un oggetto:

```
Person peterPan = g.fromJson(json, Person.class);
```

- Alcuni punti di forza:
  - Fornisce dei metodi semplici e facili da usare per «conversione» Java-JSON e viceversa
  - Genera output JSON compatti e leggibili
  - Consente rappresentazioni custom per gli oggetti
  - Consente la conversione da/a JSON di oggetti Java immodificabili pre-esistenti (non occorre modificare sorgente)
  - Supporta oggetti di complessità arbitraria

REACT

- React.js è una libreria javascript per la creazione di interfacce utente Web, rientra tra gli strumenti utili per il cosiddetto sviluppo "Front-end" di Web application
- Vocazione specifica: diventare la soluzione semplice, intuitiva e definitiva per gli sviluppatori front-end e app mobile basate su HTML5
- Libreria costruita sul linguaggio javascript, pertanto qualsiasi codice scritto in React.js esegue

all interno del browser
→ Ne consegue che React.js NON è uno strumento per lo sviluppo lato back-end delle Web application
- Come altre tecnologie front-end, React.js permette di invocare anche API lato server. React NON interagisce direttamente con database o qualsiasi altra sorgente dati che si trovi su back-end
- È in grado di interagire con tecnologie di backend quali Python/Flask, Ruby on Rails, Java/Spring, PHP, etc.
- React.js si ispira alla metodologia di sviluppo delle interfacce utenti del tipo "Single Page Application (SPA)"
    - Una SPA è un'applicazione Web che interagisce col browser per modificare pagine Web in modo dinamico in funzione dei dati che arrivano dal back-end
    - Si contrappone all'approccio classico in cui il browser carica nuove pagine in seguito all'interazione dell'utente
    - Si dice infatti che la SPA è un contenitore all'interno del quale la pagina Web evolve dinamicamente
    - Lo sviluppo di una pagina Web avviene attraverso la scrittura di cosiddetti "componenti" i quali interagiscono con le API della libreria React.js che, a loro volta, manipolano il DOM per creazione di elementi di interfaccia utente
- React.js ha introdotto **Virtual DOM**
    - Al verificarsi di un evento, invece di manipolare il DOM del browser, React.js manipola un virtual DOM che è una copia esatta del DOM del browser e si trova in memoria centrale
    - La manipolazione del Virtual DOM è più "leggera" di quella del DOM del browser
    - Lavorando con il Virtual DOM, React.js sarà in grado di inviare al DOM del browser solo le modifiche strettamente necessarie, rendendo così più leggero, efficiente e veloce il processo di rendering della pagina
    → es: se ho 10 checkbox e l'utente clicca su una, viene inviata modifica solo di quella (non ricarico tutte e 10)

Vantaggi

- Per lo sviluppatore
    - L'approccio a componenti , oltre che abilitare il riuso, permette allo sviluppatore di costruire interfacce complesse attraverso la composizione di semplici "mattoncini" (appunto, i componenti)
    - Lo sviluppatore definisce la logica dei componenti e la loro collocazione all'interno dell'interfaccia utente. La gestione del Virtual DOM, delle sue trasformazioni, della comunicazione con il DOM del browser è completamente a carico di React.js
- Per l'utilizzatore dell'interfaccia
    - L'impiego del Virtual DOM alleggerisce il processo di rendering dell'interfaccia sul browser (rendering selettivo), con un conseguente aumento delle prestazioni percettibili dall'utilizzatore

- bisogna importare librerie React:
```
<script src="https://unpkg.com/react@15/dist/react.js"></script>
<script src="https://unpkg.com/react-dom@15/dist/react-dom.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js"></script>
```

React element

- Un React element è un oggetto semplice e immutabile che descrive cosa si vuole visualizzare sullo schermo
- Solitamente è un "nodo" html (completo di eventuali nodi figli e di attributi), ma può anche avere al suo interno istanze di componenti

JSX

- Attenzione a questa istruzione:
  const elem = Hello React!
  - In questo esempio, è stata utilizzata la sintassi JSX (Javascript XML) per rappresentare in modo semplice proprio l'oggetto da visualizzare
- sintassi particolare che permette di mescolare javascript e html
- JSX permette allo sviluppatore di scrivere facilmente tag HTML all'interno di codice JavaScript e di piazzarli all'interno del DOM senza l'uso di metodi quali createElement() e/o appendChild()
- Attenzione:
  - L'uso di JSX non è obbligatorio, ma di sicuro semplifica la vita dello sviluppatore
  - React mette comunque a disposizione delle funzioni per creare elementi HTML
- Con l'impiego di JSX:

```
const myelement = <h1>I Love JSX!</h1>;
ReactDOM.render(myelement, document.getElementById('root'));
```

- Senza l'impiego di JSX:

```
const myelement = React.createElement('h1', {}, 'I do not use JSX!');
ReactDOM.render(myelement, document.getElementById('root'));
```

- Il risultato finale è identico
- il browser non è in grado di interpretare nativamente costrutti scritti in JSX, in quanto non scritti in linguaggio JavaScript → necessario aggiungere all'interno della pagina un riferimento a un cosiddetto pre-compilatore in grado di trasformare JSX in linguaggio javascript
  - Babel = compilatore che supporta traduzione in JavaScript di codice in altri linguaggi, tra cui appunto JSX

React Component

- I React Component sono oggetti complessi e dinamici, che ricevono input dall'esterno (interazioni utente, time-out, comandi dal back-end) e forgiano l'elemento grafico da restituire
- Concettualmente, i React Component sono come funzioni JavaScript: possono accettare in input dati arbitrari (sotto il nome di "props") e restituiscono elementi React che descrivono che cosa dovrebbe apparire sullo schermo
- I React Components sono i "mattoncini" fondamentali che consentono di passare da una pagina statica a un'applicazione Web dinamica la cui interfaccia è in grado di rispondere agli eventi che si verificano nella pagina, ossia reagire (react) e aggiornare se stessa di conseguenza
- Ogni "mattoncino" ha un ruolo ben definito dal punto di vista di ciò che rappresenta graficamente e si fa carico di gestire le interazioni dell'utente su quella particolare sezione di interfaccia
- In React.js i componenti sono pezzi di codice indipendenti e riusabili, richiamano il concetto di funzioni in javascript
- Esistono due tipi di componenti:

- Componenti di tipo "function"
- Componenti di tipo "class" →caratteristiche aggiuntive rispetto ai componenti di tipo "function"

- Entrambi i tipi di componenti restituiscono codice HTML attraverso l'istruzione return
- Ricordarsi che il nome del componente (anche per un componente di tipo class) deve cominciare con una lettera maiuscola, altrimenti React lo tratterebbe come un normale tag HTML
- Vincolo: la funzione deve restituire l'elemento di cui fare rendering attraverso la parola chiave return

```
function Car() {
  return <h2>I am a Car!</h2>;
}

ReactDOM.render(<Car />, document.getElementById('root'));
```

Il nome del componente va scritto qui come fosse un tag html

Il contenitore dentro cui mostrare il componente

- ReactDOM.render è l'istruzione che attiva la manipolazione del DOM e il successivo rendering su browser. In questo caso, viene impartito al DOM il comando di renderizzare il componente "Car" all'interno del contenitore div avente come id "root"
- Per creare un componente di tipo class, occorre creare una classe che estenda da React.Component e implementi obbligatoriamente il metodo render(). Così come per i componenti di tipo function, occorre che questo metodo restituisca l'elemento da renderizzare attraverso la parola chiave return

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}

ReactDOM.render(<Car />, document.getElementById('root'));
```

- React.createClass è l'istruzione per la creazione "al volo" del componente di tipo class

```
var Car = React.createClass({
    render: function() {
        return <h2>Hi, I am a Car!</h2>;
    }
});
```

- Se ho componenti annidati è sufficiente fare il rendering del componente contenitore. React, in cascata, farà il rendering degli eventuali componenti contenuti

L'oggetto props viene passato come parametro alla funzione

```
function Car(props) {
    return <h2>I am a {props.colore} Car!</h2>;
}
```

Qui viene impiegata la proprietà di nome *colore* (notazione puntata)

Al DOM viene passata una istanza della funzione col parametro colore impostato a *red*

```
ReactDOM.render(<Car colore="red"/>,
                document.getElementById('root'));
```

Props

- Sia per le funzioni che per le classi è possibile specificare delle proprietà (props = properties) ed assegnarvi valori
- In React, le props assumono valori immutabili per i quali non è prevista alcuna alterazione, utili ad esempio per configurare il componente
- L'oggetto che contiene queste proprietà prende il nome di props (parola chiave riservata)
- In fase di rendering, è possibile accedere alle props di un componente richiamandole come fossero attributi di un tag HTML
- Per le classi, l'oggetto props è built-in. Per invocarne l'uso, occorre servirsi della parola chiave

● Per le classi, l'oggetto props è built-in. Per invocarne l'uso, occorre servirsi della parola chiave "this"

State

● Tutti i componenti di tipo classe possiedono un oggetto incorporato (built-in) che prende il nome di state
  ○ A differenza delle props, le proprietà definite all'interno l'oggetto state NON sono immutabili. State è pensato proprio per contenere proprietà che nel tempo cambieranno (in seguito al verificarsi di determinati eventi)
  ○ Quando una proprietà all'interno di state cambia valore, invocata la rirenderizzazione del relativo componente

→ N.B.: componenti di tipo function sono stateless, ovvero non hanno un oggetto state

● Analogamente a tutti i linguaggi ad oggetti, anche per il tipo di componente class è possibile definire un costruttore
  ○ Viene invocato prima del rendering e funge da inizializzatore delle proprietà del componente:
    ■ Inizializzare lo stato del componente

```
class Car extends React.Component {

  constructor() {
      super();
      this.state = {color: "red"};
  }

  render() {
    return <h2>I am a {this.state.color} Car!</h2>;
  }
}
ReactDOM.render(<Car />,
                document.getElementById('root'));
```

L'oggetto state è *built-in:* per invocarne l'uso, occorre servirsi della parola chiave "this"

L'utilizzo delle proprietà di state avviene dentro la funzione render della classe

    ■ Inizializzare la gestione degli eventi
  ○ se non scrivo un constructor viene usato quello della super
  ○ Nel costruttore è stato usato il metodo super() per invocare il costruttore dell'oggetto padre e inizializzare correttamente il componente. Se non si invoca il metodo super(), non è possibile usare la keyword this all'interno del costruttore
  ○ È possibile imbattersi anche in questa sintassi (passaggio di props al costruttore):

```
constructor(props) {
    super(props);
    this.state = {.....};
}
```

● L'oggetto state di un componente classe può essere modificato attraverso la funzione setState(), che viene definita nella classe React.Component e quindi viene ereditata dai componenti classe
● L'invocazione di tale funzione scatena una reazione da parte della libreria React, la quale provvederà a modificare lo stato e a re-invocare la funzione render() della classe in modo del tutto trasparente
● Attenzione: l'oggetto state è incapsulato all'interno di un componente, il quale è l'unico ad avere diritto e responsabilità di modificarlo → Nessun altro componente potrà mai invocarne la funzione setState()
● Non tutti i componenti dovranno avere uno state, al contrario è consigliato costruire componenti senza stato (stateless)
● La tipica applicazione React è realizzata come una gerarchia di componenti: di solito, ci sono alcuni componenti ai vertici responsabili di mantenere lo stato dell'applicazione e di passare le informazioni ai componenti figli tramite props

Gestione degli eventi

- L'event handler viene solitamente realizzato attraverso un metodo della classe (in questo caso, attivaLasers)

```
class App extends React.Component {
 constructor(props) {
  super(props);
 }
 handleClick() {
  console.log("Pulsante premuto - Evento click");
 }

 render() {
  return (
   <button onClick={this.handleClick} >Pulsante</button>
  )
 }
}
```

Handler dell'evento

Stampato nella console dello strumento di ispezione del browser

```
class App extends React.Component {
 constructor(props) {
     super(props);
 }
 handleClick(e) {
     console.log('Pulsante premuto - Evento ${e.type}');
 }
 render() {
  return (
     <button onClick={this.handleClick} >Pulsante</button>
  )
 }
}
```

- il parametro e è un evento sintetico (synthetic event)
- React definisce questi eventi sintetici in base alle specifiche W3C, quindi la compatibilità tra browser è garantita
- Gli eventi React non funzionano esattamente allo stesso modo degli eventi nativi. È possibile consultare la guida di riferimento1 SyntheticEvent per saperne di più
- ATTENZIONE: Se l'handler dell'evento deve fare accesso allo state del componente, occorre apportare accorgimenti al codice di gestione dell'evento

apportare accorgimenti al codice di gestione dell'evento

- Per una corretta invocazione dell'handler dell'evento, occorre che l'oggetto invocante sia il componente. A tal fine, faremo ricorso alla keyword this
  - Ci sono due alternative:
    - All'interno del costruttore, forzare bind di this del metodo a this del componente
    - Invocare l'handler come una arrow function

```
class Interruttore extends React.Component {
  constructor(props) {
  super(props);
  this.state = {acceso: true};

  this.handleClick =
     this.handleClick.bind(this);
  }
  handleClick() {
  this.setState({acceso: !this.state.acceso}
// in alternativa
// this.setState(state => ({
//     acceso: !state.acceso
// }));
  }
  .....
```

Con questa istruzione, al 'this' dell'handler viene assegnato il 'this' del componente

```
  ........
  render() {
    return (
      <button onClick={this.handleClick}>
       {this.state.acceso ? 'Acceso' : 'Spento'}
      </button>
    );
  }
}
```

Event handler invocato come stringa

```
class Interruttore extends
   React.Component {
   constructor(props) {
  super(props);
  this.state = {acceso: true};

 handleClick() {
  this.setState({acceso:
   !this.state.acceso}
  // in alternativa
  //  this.setState(state => ({
  //      acceso: !state.acceso
  // }));
 }
.....
```

```
  ........
  render() {
   return (
     <button onClick = {() = >
    this.handleClick()}>
      {this.state.acceso ? 'Acceso' :
    'Spento'}
     </button>
   );
  }
 }
```

Invocazione dell'handler tramite arrow function

Form

- Gli elementi form mantengono naturalmente uno stato interno
- Gli elementi di un form quali <input>, <textarea> e <select> mantengono e aggiornano il proprio stato in base all'input dell'utente
- In React, come è già noto, lo stato mutabile viene mantenuto nella proprietà state del componente e viene poi aggiornato solo mediante setState()
- Bisogna usare gestione eventi → evento onChange / evento onSubmit

- In React si possono effettuare HTTP request in diversi modi
- Uno elegante (e dal semplice utilizzo) fa uso delle Fetch API fornite nativamente da javascript
  - Le Fetch API forniscono un'interfaccia js per accedere e manipolare parti della pipeline HTTP, come ad es. richieste e risposte
  - Mettono a disposizione, inoltre, un metodo che fornisce un modo semplice e logico per recuperare le risorse in modo asincrono

Librerie e Framework alternativi a React.Js

- Oltre a React.js esistono numerose iniziative che propongono librerie e framework basate su javascript
- L'obiettivo di ciascuna iniziativa è quello di fornire allo sviluppatore uno strumento/ambiente di sviluppo lato front-end più "comodo" rispetto a javascript (soprattutto per la gestione del DOM) e che possa abbattere i tempi di sviluppo delle interfacce delle applicazioni Web

jQuery

jQuery

- La libreria opensource jQuery è in assoluto la più utilizzata e conosciuta dalla comunità degli sviluppatori
- jQuery semplifica molto la gestione degli elementi DOM e presenta diverse funzioni per questo scopo: con i selettori del CSS3 si possono selezionare facilmente e manipolare gli elementi della pagina. Inoltre, offre una gestione semplificata delle richieste Ajax
- Il codice è compatibile con tutti i browser ed esistono molti plug-in ■ È una componente essenziale di molti CMS come WordPress, Drupal o Joomla!
- La sua estension jQuery UI è particolarmente adatta per realizzare effetti semplici ed elementi interattivi come drag&drop, ingrandimento e ridimensionamento degli elementi del sito, animazioni ed effetti vari

Angular

- Creato e mantenuto da Google, è il successore di AngularJS. Insieme a React.js, dispone di una grande community di sviluppatori
- È riconosciuto come l'antagonista principale di React.js. Analogamente a React.js, serve per realizzare Single Page Application
- Implementa il design pattern MVVM (Model View ViewModel). Si basa su jQuery Lite, una variante compatta della altrettanto famosa libreria js jQuery
- Rispetto al suo antecedente (AngularJS) la differenza principale è che per la programmazione non viene più utilizzato JavaScript, ma TypeScript, un linguaggio di programmazione sviluppato da Microsoft che si basa su javascript
- Punto di forza è la facilità di sviluppo delle applicazioni per diversi dispositivi (desktop, mobile, tablet)

Vue.js

- Analogamente ad Angular e React, Vue.js è un framework js per lo sviluppo di Single Page Application
- Adotta il design pattern Model-View-ViewModel
- L'intento degli sviluppatori è stato creare uno strumento più facile per i principianti rispetto agli altri framework
- Ciò, però, va a discapito della completezza di funzionalità (in cui i competitor eccellono), per le quali però è comunque possibile integrare un numero ristretto di librerie aggiuntive opzionali

Meteor

- Meteor, o MeteorJS, è un framework javascript particolarmente adatto per lo sviluppo su diverse piattaforme
- Consente agli sviluppatori di creare con lo stesso codice sia applicazioni Web sia app per i dispositivi mobili
- Un altro vantaggio consiste nel fatto che le modifiche al codice possono essere inoltrate direttamente ai client grazie al protocollo proprietario Distributed Data Protocol (DDP)
- Funziona su una base Node.js, pertanto può essere impiegato sia per sviluppo front-end che per sviluppo back-end
- Risulta molto utile disporre di conoscenze su Node.js per lavorare con Meteor

Backbones

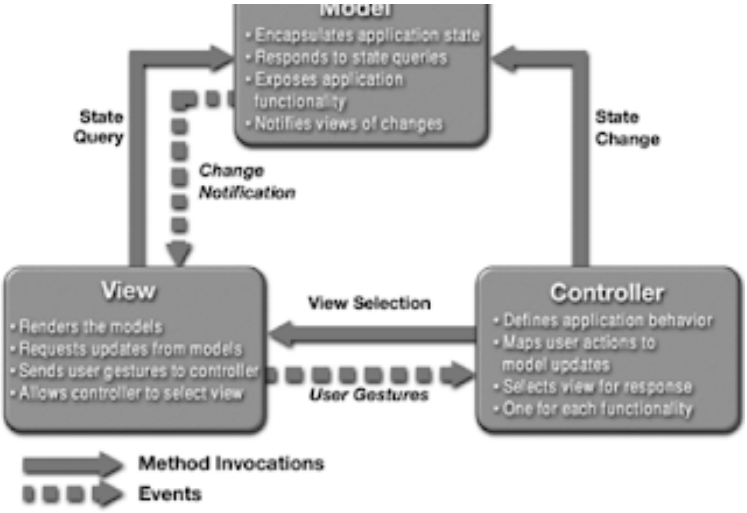
- Backbones non è un vero e proprio framework ma, piuttosto, un ottimo strumento per modellare e strutturare il codice
- Grazie a questa caratteristica, backbones lascia più spazio al programmatore. Per contro, impiegato da solo non fornisce un framework completo, quindi lo si deve abbinare obbligatoriamente ad altre librerie quali underscore.js e jquery
- È nato per sviluppare applicazioni single-page ed adotta il design pattern Model-View-Presenter (MVP)

JAVA MODEL 2

- Nel progetto di applicazioni Web in Java, due modelli di ampio uso e di riferimento: Model 1 e Model 2
    - ○ Model 1 è un pattern semplice in cui codice responsabile per presentazione contenuti è mescolato con logica di business → suggerito solo per piccole applicazioni (sta diventando obsoleto nella pratica industriale)
    - ○ Model 2 design pattern più complesso e articolato che separa chiaramente livello presentazione dei contenuti dalla logica utilizzata per manipolare e processare contenuti stessi → suggerito per applicazioni di medio-grandi dimensioni
- Visto che Model 2 preme per separazione netta fra logica di business e di presentazione, è usualmente associato con paradigma Model-View-Controller (MVC)
    - ○ Mai specificato in modo definitivo e "mandatory" come realizzare tecnicamente modello MVC in tecnologie Java
    → Battaglia delle implementazioni: diverse soluzioni possibili (ad es. Java BluePrints suggerisce adozione di Enterprise Java Bean - EJB - per realizzare modello MVC)
    - ○ Proposta di separazione logica di business (servlet) da presentazione (JSP), con le due parti viste come "Controller" e "View" rispettivamente
    - ○ Parte di "Model" lasciata non specificata nell'architettura proposta da Govind (idea che ogni struttura dati possa essere adatta a realizzare modello: da Vector list a db relazionale)
    - ○ Spesso oggi due termini Model 2 e MVC vengono (impropriamente) usati come sinonimi fra gli sviluppatori

Architettura Model View Controller

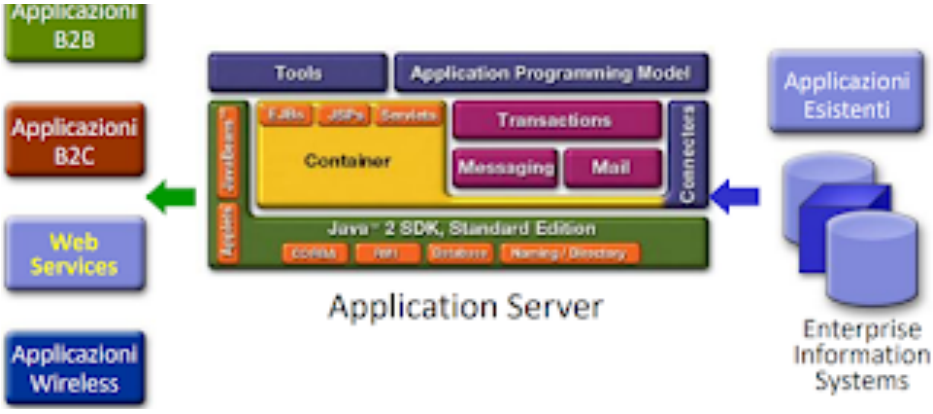
- Architettura adatta per applicazioni Web interattive (altre tecnologie e modelli più adatti per servizi asincroni, vedi Message Driven Bean e Java Messaging Service - JMS)
- Model - rappresenta livello dei dati, incluse operazioni per accesso e modifica. Model deve notificare view associate quando modello viene modificato e deve supportare:
    - ○ possibilità per view di interrogare stato di model
    - ○ possibilità per controller di accedere alle funzionalità incapsulate da model
- View - si occupa di rendering dei contenuti di model. Accede ai dati tramite model e specifica come dati debbano essere presentati
    - ○ aggiorna presentazione dati quando modello cambia
    - ○ gira input utente verso controller
- Controller - definisce comportamento dell'applicazione
    - ○ fa dispatching di richieste utente e seleziona view per presentazione
    - ○ interpreta input utente e lo mappa su azioni che devono essere eseguite da model (in una GUI stand-alone, input come click e selezione menu; in una applicazione Web, richieste HTTP GET/POST)
- Mapping possibile su applicazioni Web Java-based In applicazioni Web conformi a Model 2, richieste del browser cliente vengono passate a controller (usualmente implementato da servlet oppure EJB Session Bean oppure...)
- Controller si occupa di eseguire logica business necessaria per ottenere il contenuto da mostrare. Controller mette il contenuto (usualmente sotto forma di JavaBean o Plain Old Java Object - POJO) in messaggio e decide a quale view (usualmente implementata da JSP) passare la richiesta
- View si occupa del rendering contenuto (ad es. stampa dei valori contenuti in struttura dati o bean, ma anche operazioni più complesse come invocazione metodi per ottenere dati)
- Model-View-Controller (architettura generale)



- Eventi e invocazioni
- In Java Model 2, tipicamente controller come EJB Session Bean o servlet, e view come JSP

- architetture multi-tier:
  - Complessità del middle tier server
  - Duplicazione dei servizi di sistema per la maggior parte delle applicazioni enterprise
  - Controllo concorrenza, transazioni
  - Load-balancing, sicurezza
  - Gestione risorse, connection pooling
- Modo per risolvere il problema:
  - Container condiviso che gestisce i servizi di sistema → Proprietario vs. basato su standard aperti?
- Soluzioni proprietarie:
  - Usano il modello componente-container
  - Componenti per la business logic
  - Container per fornire servizi di sistema
  - Il contratto componenti-container è ben definito, ma in modo proprietario (problema di vendor lock-in)
  - Esempi: Tuxedo, .NET
- Soluzioni basate su standard aperti
  - Usano il modello componente-container e il container fornisce i servizi di sistema in modo ben definito in accordo a standard industriali
  - Ad es. J2EE e Java Specification Request (JSR) (tra l'altro, anche supporto a portabilità di codice perché basato su bytecode Java e API di programmazione definite in standard aperti)

- Application server Java Enterprise Edition = piattaforma open e standard per lo sviluppo, il deployment e la gestione di applicazioni enterprise n-tier, Web-enabled, server-centric e basate su componenti



J2EE per Applicazioni N-tier

- *Modello 4-tier e applicazioni J2EE*
  - Cliente HTML, JSP/Servlet, EJB, JDBC/Connector
- *Modello 3-tier e applicazioni J2EE*
  - Cliente HTML, JSP/Servlet, JDBC
- Modello 3-tier e applicazioni J2EE
  - Applicazioni standalone EJB client-side, EJB, JDBC/Connector
- Applicazioni enterprise B2B
  Interazioni tra piatta-forme J2EE tramite messaggi JMS o XML-based

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
- ...

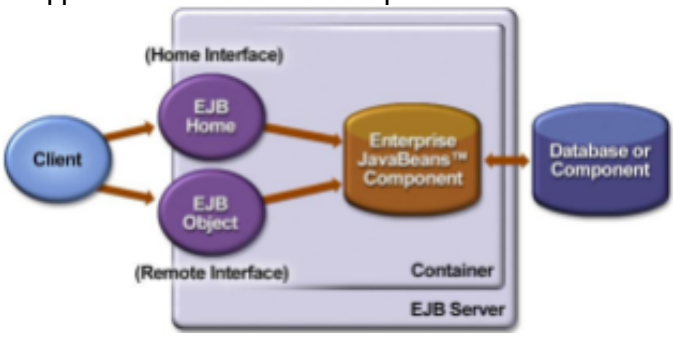
EJB

- Tecnologia per componenti server-side
- Sviluppo e deployment semplificato di applicazioni Java
  - → Distribuite, con supporto alle transazioni, multi-tier, portabili, scalabili, sicure, ...
- Porta a semplifica i benefici del modello a componenti sul lato server

- Porta e amplifica i benefici del modello a componenti sul lato server
- Separazione fra logica di business e codice di sistema
- Container per la fornitura dei servizi di sistema
- Modello a container pesante (contrapposto a possibili modelli alternativi, come container leggero Spring)
- Rende possibile (e semplice) la configurazione a deployment-time → Deployment descriptor

Principi di Design

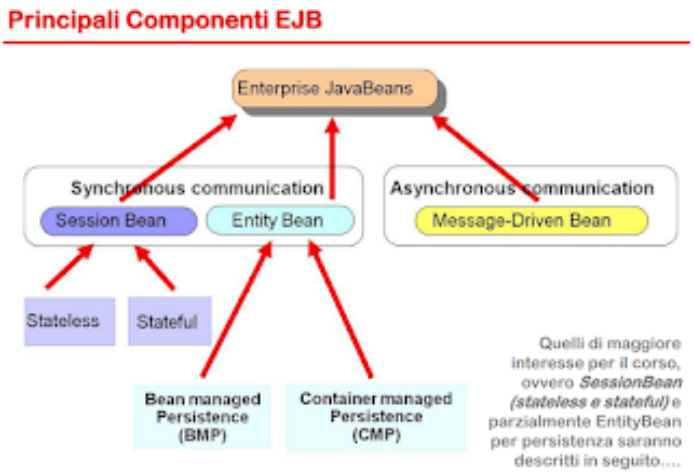
- Applicazioni EJB e i loro componenti devono essere debolmente accoppiati (loosely coupled)



- Comportamento di EJB definito tramite interfacce
- Applicazioni EJB NON si occupano della gestione delle risorse
- Applicazioni EJB sono N-tier
    - Session tier come API verso l'applicazione
    - Entity tier come API verso le sorgenti dati
- Idea di base: container pesante attivo all'interno di un EJB Server (Application Server)
- Cliente può interagire remotamente con componente EJB tramite interfacce ben definite passando SEMPRE attraverso container

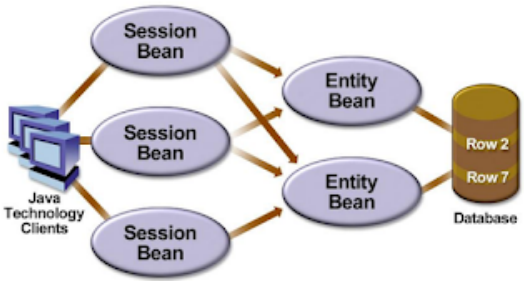
Descrittori di Deployment

- Forniscono istruzioni al container su come gestire e controllare il comportamento (anche runtime) di componenti J2EE
    - Transazioni
    - Sicurezza
    - Persistenza
    - ...
- Permettono la personalizzazione tramite specifica dichiarativa (NO personalizzazione tramite programmazione)
- Semplificano portabilità del codice
- Sostituiti o sostituibili con annotazioni a partire da Java5



Componenti Bean

- Session Bean (stateless / stateful)
  - Lavorano tipicamente per un singolo cliente
  - Non sono persistenti (vita media relativamente breve)
  - Persi in caso di failure di EJB server
  - Non rappresentano dati in un DB, anche se possono accedere/modificare questi dati
  - In EJB2.x classe Bean deve implementare interfaccia javax.ejb.SessionBean; in EJB3.x solo uso di annotazioni
  - Uso:
    - modellare oggetti di processo o di controllo specifici per un particolare cliente
    - modellare workflow o attività di gestione e per coordinare interazioni fra bean
    - muovere la logica applicativa di business dal lato cliente a quello servitore
  - Stateless: restituisce risultato senza salvare alcuna informazione di stato relativa al cliente: transienti, elemento temporaneo di business logic necessario per specifico cliente



per intervallo di tempo limitato
  - Stateful: può mantenere stato specifico per un cliente
- Entity Bean
  - Forniscono una vista ad oggetti dei dati mantenuti in un database
    - Tempo di vita non connesso alla durata delle interazioni con i clienti
    - Componenti permangono nel sistema fino a che i dati esistono nel database - long lived
    - Nella maggior parte dei casi, componenti sincronizzati con relativi database relazionali
  - Accesso condiviso per clienti differenti
  - In EJB2.x classe Bean deve implementare interfaccia javax.ejb.EntityBean; in EJB3.x supporto alla persistenza simile a Hibernate

Session Bean	Entity Bean
Rappresenta un processo di business	Rappresenta dati di business
Una istanza per cliente	Istanza condivisa fra clienti multipli
Short-lived: vita del bean pari alla vita cliente	Long-lived: vita del bean pari a quella dei dati nel database
Transient	Persistente
Non sopravvive a crash del server	Sopravvive a crash del server
Può avere proprietà transazionali	Sempre transazionale

- Message-Driven Bean (MDB)
  - Svolgono il ruolo di consumatori di messaggi asincroni → perché è importante che siano asincroni?

- Non possono essere invocati direttamente dai clienti
- Attivati in seguito all'arrivo di un messaggio
- I clienti possono interagire con MDB tramite l'invio di messaggi verso le code o i topic per i quali questi componenti sono in ascolto (listener)
- Privi di stato
- Nel caso di utilizzo di JMS
    - MDB corrispondente deve implementare l'interfaccia javax.jms.MessageListener interface
    - L'implementazione del metodo onMessage() deve contenere la business logic
    - Il bean viene configurato come listener per queue o topic JMS

Tipologie di Bean EJB3.0

- bean di tipo sessione e message-driven sono classi Java ordinarie (Plain Old Java Object - POJO)
    - Rimossi i requisiti di interfaccia
    - Tipo di bean specificato da una annotation (o da un descrittore)
    - Annotation principali: @Stateless, @Stateful, @MessageDriven
    - Specificati nella classe del bean
    - Gli entity bean di EJB2.x non sono stati modificati e possono continuare a essere utilizzati
        - Ma Java Persistence API supporta nuove funzionalità
        - @Entity si applica solo alle nuove entità relative a Java Persistence API

Servizi Container-based

- quali servizi di supporto/sistema e come vengono supportati in un modello a container pesante?
    - Pooling e concorrenza
    - Transazionalità
    - Gestione delle connessioni a risorse
    - Persistenza (vedi Java Persistence API - JPA - e supporto Hibernate ORM)
    - Messaggistica (vedi Java Messaging System - JMS)
    - (Sicurezza)

Gestione della concorrenza

- Resource Pooling: Pooling dei componenti server-side da parte di EJB container (**instance pooling**). Idea base è di evitare di mantenere istanza separata di ogni EJB per ogni cliente. Si applica a stateless session bean e message-driven bean (che cos'è un cliente per un MDB?) → Anche pooling dei connector
- Activation: Utilizzata da stateful session bean per risparmiare risorse
- Session bean non possono essere concorrenti, nel senso che una singola istanza è associata ad un singolo cliente
    - Vietato l'utilizzo di thread a livello applicativo e, ad esempio, della keyword synchronized
    - Come nel caso di stateless session bean, Message Driven Bean non mantengono stato della sessione e quindi container può effettuare pooling in modo relativamente semplice
    - Strategie di pooling analoghe alle precedenti. Unica differenza che ogni EJB container contiene molti pool, ciascuno dei quali è composto di istanze con la stessa destination JMS
- **Stateless Session Bean**
    - Ogni EJB container mantiene un insieme di istanze del bean pronte per servire richieste cliente
    - Non esiste stato di sessione da mantenere fra richieste successive; ogni invocazione di metodo è indipendente dalle precedenti Implementazione delle strategie di instance pooling demandate ai vendor di EJB container, ma analoghi principi
    - Ciclo di vita di uno stateless session bean:
        - No state (non istanziato; stato iniziale e terminale del ciclo di vita)
        - Pooled state (istanziato ma non ancora associato ad alcuna richiesta cliente)

        - Pooled state (istanziato ma non ancora associato ad alcuna richiesta cliente)
        - Ready state (già associato con una richiesta EJB e pronto a rispondere ad una invocazione di metodo)
    - Istanza del bean nel pool riceve un riferimento a javax.ejb.EJBContext (in caso di richiesta di injection nel codice tramite apposita annotation) → EJBContext fornisce interfaccia per il bean per comunicare con ambiente EJB
    - Quando il bean passa in stato ready, EJBContext contiene anche informazioni sul cliente che sta utilizzando il bean. Inoltre contiene il riferimento al proprio EJB stub, utile per passare riferimenti ad altri bean
    - Ricordiamo che il fatto di essere stateless è indicato semplicemente tramite annotation @javax.ejb.Stateless (NOTA: variabili di istanza non possono essere usate per mantenere stato della sessione)
- **Stateful Session Bean: Activation**
    - Usata nel caso di stateful session bean
    - Gestione della coppia oggetto EJB + istanza di bean stateful
    - Passivation: disassociazione fra stateful bean instance e suo oggetto EJB, con salvataggio dell'istanza su memoria (serializzazione). Processo del tutto trasparente per cliente
    - Activation: recupero dalla memoria (deserializzazione) dello stato dell'istanza e riassociazione con oggetto EJB
    - Nella specifica J2EE, non richiesto che classe di uno stateful session bean sia serializzabile. → Dipendenza dall'implementazione dello specifico vendor e attenzione al trattamento dei transient…
    - La procedura di activation può essere associata anche all'invocazione di metodi di callback sui cambi di stato nel ciclo di vita di uno stateful session bean.
        - Esempio: annotation @javax.ejb.PostActivate associa invocazione del metodo a cui si applica immediatamente dopo l'attivazione di un'istanza
        - Similmente, @javax.ejb.PrePassivate (prima dell'azione di passivation)
        - Utilizzati spesso per la chiusura/apertura di connessioni a risorse per gestione più efficiente

Spring

- Modello a container leggero per costruire applicazioni Java SE e Java EE
- Nasce come alternativa non ufficiale a ejb
- Prima tecnologia java oriented con Inversion of Control (IoC) e Dependency Injection
- Altre funzionalità: supporto alla persistenza, integrazione con Web tier, Aspect Oriented Programming (AOP)
- Spring rappresenta un approccio piuttosto unico (che ha fortemente influenzato i container successivi, verso tecnologie a microcontainer)
- Proprietà originali:
    - Spring come framework modulare. Architettura a layer, possibilità di utilizzare anche solo alcune parti in isolamento → Anche possibilità di introdurre Spring incrementalmente in progetti esistenti e di imparare a utilizzare la tecnologia "pezzo per pezzo"
    - Supporto a importanti aree non coperte da altri framework diffusi, come la gestione degli oggetti di business
    - Tecnologia di integrazione di soluzioni esistenti
    - Facilità di testing

- Gestione della configurazione dei componenti applica principi di Inversion-of-Control e utilizza Dependency Injection

→ Eliminazione della necessità di binding "manuale" fra componenti

Eliminazione della necessità di binding "manuale" fra componenti

- Idea fondamentale di una factory per componenti (BeanFactory) utilizzabile globalmente. Si occupa fondamentalmente del ritrovamento di oggetti per nome e della gestione delle relazioni fra oggetti (configuration management)
- Persistenza
  - Livello di astrazione generico per transazioni con DB (senza essere forzati a lavorare dentro EJB container)
  - Integrazione con framework di persistenza come Hibernate, JDO, JPA

Web tier e AOP

- Integrazione con Web tier
  - Framework MVC per applicazioni Web, costruito su funzionalità base Spring, con supporto per diverse tecnologie per generazione viste, es: JSP, FreeMarker, Velocity, Tiles, iText, POI (Java API per accesso a file in formato MS)
  - Web Flow per navigazione a grana fine
- Supporto ad Aspect Oriented Programming
  - Framework di supporto a servizi di sistema, come gestione delle transazioni, tramite tecniche AOP
    - Miglioramento soprattutto in termini di modularità
    - Parzialmente correlata anche la facilità di testing

Moduli Spring d'interesse

- Core Package
  - Parte fondamentale del framework. Consiste in un container leggero che si occupa di Inversion of Control o, per dirla alla Fowler, Dependency Injection
  - L'elemento fondamentale è BeanFactory, che fornisce una implementazione estesa del pattern factory ed elimina la necessità di gestione di singleton a livello di programmazione, permettendo di disaccoppiare configurazione e dipendenze dalla logica applicativa
- MVC Package
  - Ampio supporto a progettazione e sviluppo secondo architettura Model-View-Controller (MVC) per applicazioni

Dependency Injection

- Applicazione più nota e di maggiore successo del principio di Inversion of Control
  → "Hollywood Principle": Don't call me, I'll call you
  (se attore chiama il suo agente ogni giorno per sapere se ha ruoli, agente ha telefono intasato di chiamate perché segue più di un attore → dice che lo chiamerà lui quando ci saranno parti per lui)
- Container (in realtà il container leggero di Spring) si occupa di risolvere (injection) le dipendenze dei componenti attraverso l'opportuna configurazione dell'implementazione dell'oggetto (push)
- Opposta ai pattern più classici di istanziazione di componenti o Service Locator (→ sa dove trovare risorse e servizi necessari), dove è il componente che deve determinare l'implementazione della risorsa desiderata (pull)

→ non cerco io risorsa, è il framework che mi indirizza (INIETTA) → codice più puro e indipendente da risorsa, flessibile per cambiamento, mantenibile, più facilità di testing

→ es. io chiamo printer.print poi è il container che sa indirizzo logico stampante (se la cambio non devo cambiare codice di lookup ma solo configurazione)

- Benefici:
  - Flessibilità → Eliminazione di codice di lookup nella logica di business
  - Possibilità e facilità di testing → Nessun bisogno di dipendere da risorse esterne o da container in fase di testing e possibilità di abilitare testing automatico
  - Manutenibilità → Permette riutilizzo in diversi ambienti applicativi cambiando semplicemente i file di configurazione (o in generale le specifiche di dependency injection) e

non il codice

- Supporto a dependency injection si occupa di creare oggetti/componenti quando necessario e di passarli automaticamente agli oggetti/componenti che li devono utilizzare → Idea base per implementazione: costruttori in oggetto A che accettano B e C come parametri di ingresso, oppure A contiene metodi setter che accettano interfacce B e C come parametri di inqresso

❑ Dependency injection *a livello di costruttore*

Dipendenze fornite attraverso i costruttori dei componenti

```
public class ConstructorInjection {
    private Dependency dep;
    public ConstructorInjection(Dependency dep) {
        this.dep = dep;  } }
```

❑ Dependency injection *a livello di metodi "setter"*

- Dipendenze fornite attraverso i *metodi di configurazione* (*metodi setter in stile JavaBean*) dei componenti
- Più frequentemente utilizzata nella comunità degli sviluppatori

```
public class SetterInjection {
    private Dependency dep;
    public void setMyDependency(Dependency dep) {
        this.dep = dep;    } }
```

BeanFactory

- L'oggetto BeanFactory è responsabile della gestione dei bean che usano Spring e delle loro dipendenze
- Ogni applicazione interagisce con la dependency injection di Spring (IoC container) tramite l'interfaccia BeanFactory
  - Oggetto BeanFactory viene creato dall'applicazione, tipicamente nella forma di XmlBeanFactory
  - Una volta creato, l'oggetto BeanFactory legge un file di configurazione e si occupa di fare l'injection
- XmlBeanFactory è estensione di DefaultListableBeanFactory per leggere definizioni di bean da un documento XML

```
public class XmlConfigWithBeanFactory {
    public static void main(String[] args) {
        XmlBeanFactory factory = new XmlBeanFactory(new FileSystemResource("beans.xml"));
            SomeBeanInterface b = (SomeBeanInterface) factory.
getBean("nameOftheBean");
    }
```



```
<beans>
    <bean id="renderer" class="StandardOutMessageRenderer">
        <property name="messageProvider">
            <ref local="provider"/>
        </property>
    </bean>
    <bean id="provider" class="HelloWorldMessageProvider"/>
</beans>
```

Oppure a livello di costruttore

Vedete anche l'esempio successivo, lucido 43, in cui in grande dettaglio sarà mostrato come il container IoC faccia dependency injection

```
...
<beans>
<bean id="provider" class="ConfigurableMessageProvider">
        <constructor-arg>
            <value> Questo è il messaggio configurabile</value>
        </constructor-arg>
    </bean> </beans>
```

property indica che questo componente ha una dipendenza
ref local per impostare nome logico (così poi cambi indirizzo risorsa solo da qui)

```
public class ConfigurableMessageProvider implements MessageProvider {
    private String message;
    // usa dependency injection per config. del messaggio
    public ConfigurableMessageProvider(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }
}
```