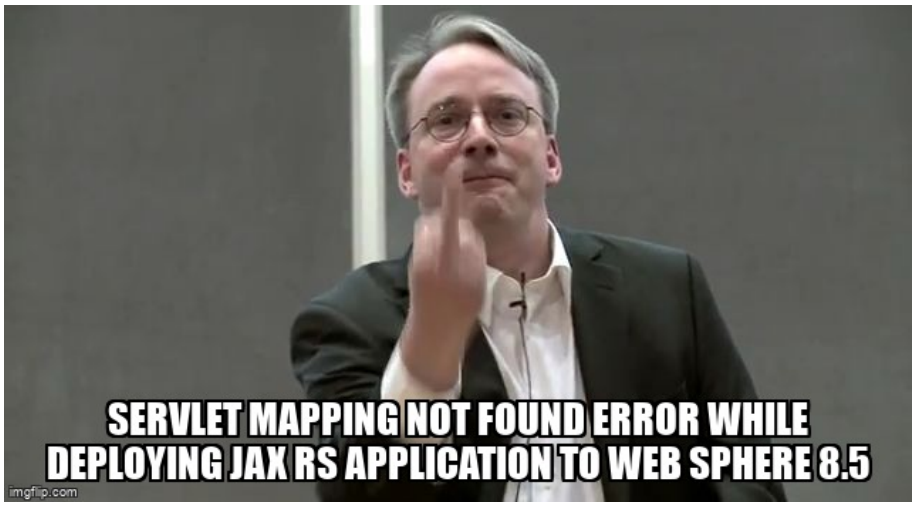
Tecnologie Web-T
Q&A



Premessa

Ho scritto questo file in modo da facilitare lo studio e il superamento dell'esame orale di Tecnologie Web-T.
Tuttavia è consigliato integrare questo materiale con le slide del professor Paolo Bellavista, disponibili sulla piattaforma Virtuale.

Buono studio :)

1 Web Statico

1.1 HTML

Domanda 1

Che cos'è un ipertesto, dove viene utilizzato e quali sono le sue componenti?

Un ipertesto è un insieme di documenti messi in relazione tra loro per mezzo di parole chiave: può essere visto come una rete con i singoli documenti che ne costituiscono i nodi.
HTML è un tipo di ipertesto in cui i vari collegamenti sono identificati dai **link**.

Domanda 2

Che cosa sono URI e URL? Qual è la loro differenza?

- **URI (Uniform Resource Identifier)**: identificatore di risorse(=una qualunque entità che abbia un'identità), un mapping concettuale di un'entità (es.edizione di un libro).
- **URL (Uniform Resource Locator)**: identifica una risorsa per mezzo del suo meccanismo di accesso primario. Specifica il protocollo necessario per il trasferimento della risorsa ed esso corrisponde al nome dello schema

La differenza fondamentale fra i due è che l'URI è un concetto generico, mentre l'URL è un caso specifico di URI che da anche informazioni riguardo alla locazione (logica o fisica) della risorsa.

Domanda 3

Cosa significa *cascade* in CSS?

Lo standard CSS definisce un insieme di regole di risoluzione dei conflitti che prende il nome di cascade. La logica di risoluzione si basa su tre elementi (origine del foglio di stile):

- **Autore**: stile definito nella pagina.
- **Browser**: foglio di stile predefinito.
- **Utente**: Modifiche apportate in locale sul browser.

Il CSS assegna un peso a ciascun blocco di regole e 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.

Domanda 4

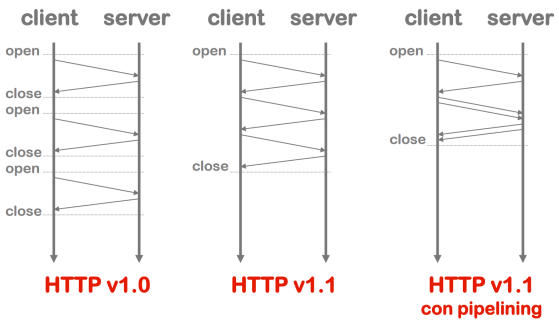
Qual è la differenza fra HTML e SGML?

SGML (Standard Generalized Markup Language) è un meccanismo flessibile e portabile per rappresentare documenti elettronici composto da vari elementi quali capitoli, titoli, riferimenti, oggetti grafici, etc., ma **non contiene sequenze di istruzioni di formattazione**.
HTML è un'applicazione/semplificazione di SGML, ovvero un linguaggio per la rappresentazione di un tipo di documento SGML. Oltre a descrivere il contenuto, **HTML associa anche significati grafici agli elementi che definisce!**

1.2 HTTP

Domanda 5

Differrenza HTTP 1.0, HTTP 1.1 e HTTP 1.1 con pipelining



HTTP 1.0 è un protocollo request-response, stateless, one-shot.
La differenza principale tra HTTP 1.0 e 1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione (le connessioni 1.0 vengono dette non persistenti mentre quelle 1.1 vengono definite persistenti). **Il server lascia aperta la connessione TCP** dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione.
Il pipelining consiste nell'invio di **molteplici richieste da parte del client prima di terminare la ricezione delle risposte**. Le risposte debbono però essere date nello stesso ordine delle richieste, poiché non è specificato un metodo esplicito di associazione tra richiesta e risposta.

Domanda 6

Differenza metodi GET e POST

Servono entrambi per **richiedere una risorsa** ad un server (senza crearne una).
Il metodo GET è il più frequente: è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser. Prevede il **passaggio di parametri nella parte <query> dell'URL**, di conseguenza in modo limitato (lunghezza massima URL: 2048 caratteri). Col metodo POST, a differenza di GET, **i dettagli** per identificazione ed elaborazione della risorsa stessa non sono nell'URL, ma **sono contenuti nel body** del messaggio, quindi senza limiti di lunghezza

Domanda 7

Altre richieste oltre GET e POST (PUT, DELETE, OPTION, HEAD, TRACE)

- **PUT**: chiede la memorizzazione sul server di una risorsa all'URL specificato. Il body messaggio del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET con lo stesso nome in seguito.
- **DELETE**: richiede la cancellazione della risorsa riferita dall'URL specificato. PUT e DELETE sono normalmente disabilitati sui server pubblici.
- **HEAD**: risponde soltanto con gli header relativi, senza body. Viene usato per verificare di un URL la sua validità (risorsa inesistente o vuota, tramite content-length) o la sua accessibilità (es. 403 Forbidden).
- **OPTIONS**: serve per richiedere informazioni sulle opzioni disponibili per la comunicazione.
- **TRACE**: è usato per invocare il loop-back remoto a livello applicativo del messaggio di richiesta. Consente al client di vedere che cosa è stato ricevuto dal server: viene usato nella diagnostica e nel testing dei servizi Web.

Domanda 8

Informazioni contenute nei cookie (campi)

I cookie sono una collezione di stringhe:

- **Key**: identifica univocamente un cookie all'interno di un dominio.
- **Value**: valore associato al cookie (è una stringa di max 255 caratteri).
- **Path**: posizione nell'albero di un sito al quale è associato (di default /).
- **Domain**: dominio dove è stato generato.

- **Max-age (opzionale)**: numero di secondi di vita (permette la scadenza di una sessione).
- **Secure (opzionale)**: utilizzo soltanto se il protocollo è sicuro (https).
- **Version**: identifica la versione del protocollo di gestione dei cookie,

Domanda 9

Come si fa caching sul web browser in HTTP?

Lo user agent (tipicamente il browser) mantiene una cache delle pagine visitate dall'utente. L'uso delle user agent cache era molto importante in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga.
Questo modello di caching è 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 a caricamento di elementi statici (icone, sfondi, ecc.).

Domanda 10

Aggiornamento proxy cache (Freshness, Validation, Invalidation)

HTTP definisce vari 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

1.3 CGI

Domanda 11

CGI (Common Gateway Interface)

CGI è uno standard per interfacciare applicazioni esterne con Web server.
Un programma CGI viene eseguito dinamicamente e può essere scritto in qualunque linguaggio. Le operazioni si svolgono nel seguente ordine:

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

I programmi CGI e il Server comunicano in 4 modi:

1. **Variabili di ambiente** del sistema operativo.
2. **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).
3. **Standard Input** (usato con il metodo POST).
4. **Standard Output**: per restituire al server la pagina HTML da inviare al client.

2 Web Dinamico

2.1 Servlet

Domanda 12

Ciclo di vita delle Servlet(normale/multithread vs deprecato/singlethread)

Il ciclo di vita di una Servlet viene controllato dal **Servlet Container** e può agire in 2 modi:

- **Normale/MultiThread**: Viene creata **una sola istanza** della classe Servlet e per ogni richiesta viene creato un thread (possibili problemi di concorrenza).
- **Deprecato/SingleThread**: Vengono create **tante istanze** della classe Servlet, una per ogni richiesta in arrivo (ognuna con un solo Thread, no problemi concorrenza).

Alla creazione di un instanza della classe Servlet viene chiamato il metodo `init()`, poi per ogni richiesta in arrivo viene chiamato il metodo `service()` che a sua volta chiama i metodi `doGet()` oppure `doPost()`.

Domanda 13

Cookie: che cos'è e come si può usare in Java lato server?

Il cookie è un'unità di informazione che il Web Server deposita sul Web Browser lato cliente. Può contenere valori che sono propri del dominio funzionale dell'applicazione (in genere informazioni associate all'utente). Possono essere rifiutati dal browser (tipicamente perché disabilitati). Vengono mandati avanti e indietro nelle richieste e nelle risposte e memorizzati dal browser *(client maintained state)*.
La classe Cookie modella il cookie HTTP. Si recuperano i cookie dalla request utilizzando il metodo `getCookies()`, e si aggiungono cookie alla response utilizzando il metodo `addCookie()`.

Domanda 14

Differenza include/forward

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).

- **Include**: la servlet inoltra una request ad un componente Web che la elabora e restituisce il risultato, che poi viene trattato come risorsa statica e aggiunto alla pagina.
- **Forward**: 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. La risposta è di competenza esclusiva della risorsa che riceve l'inoltro.

2.2 JSP

Domanda 15

Ciclo di vita di una JSP

Le richieste verso JSP sono gestite da una particolare servlet (in Tomcat si chiama JspServlet) che effettua le seguenti operazioni:

1. Traduzione della JSP in una servlet.
2. Compilazione della servlet risultante in una classe.
3. Esecuzione della JSP.

Dal momento che le JSP sono compilate in servlet, il ciclo di vita delle JSP, dopo compilazione, è controllato sempre nel medesimo modo delle servlet .

Domanda 16

Differenza JSP e servlet

Le JSP sono nate a causa dei seguenti svantaggi delle servlet:

- Generazione HTML direttamente in Java.
- Il processo di generazione della pagina è timeconsuming, ripetitivo e soggetto a errori (sequenza di `println()`).
- L'aggiornamento delle pagine è molto scomodo.

Le JSP hanno quindi i seguenti vantaggi:

- Si separa 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.

Domanda 17

Quando usare JSP e quanso usare Servlet?

Le JSP non rendono inutili le servlet. Le servlet forniscono agli sviluppatori delle applicazioni Web un completo controllo dell'applicazione.
Se si vogliono fornire contenuti differenziati a seconda di diversi parametri quali l'identità dell'utente, condizioni dipendenti dalla business logic, etc. è conveniente continuare a lavorare con le servlet.
Le 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.

Domanda 18

Come funziona la direttiva `page`?

La direttiva page serve per definire una serie di attributi che si applicano all'intera pagina.

- `lenguage="java"`: definisce il linguaggio di scripting utilizzato nelle parti dinamiche, allo stato attuale l'unico valore ammesso è `"java"`.
- `import="{package.class|package.*}"`: definisce la lista di package da importare. Gli import più comuni (quelli sulle servlet) sono impliciti e non serve inserirli.
- `session="true|false"`: indica se la pagina fa uso della sessione (altrimenti non si può usare l'oggetto `session`).
- `buffer="none|8kb|sizekb"`: indica la dimensione del buffer di uscita.
- `autoFlush="true|false"`: indica se il buffer viene svuotato automaticamente quando è pieno. Se il valore è false viene generata un'eccezione quando il buffer è pieno.
- `isThreadSafe="true|false"`: indica se il codice contenuto nella pagina è thread-safe. Se vale false le chiamate alla JSP vengono serializzate (no multithreading).
- `info="text"`: testo di commento. Può essere letto con il metodo `Servlet.getServletInfo()`.
- `errorPage="relativeURL"`: indirizzo della pagina a cui vengono inviate le eccezioni.
- `isErrorPage="true|false"`: indica se la JSP corrente è una pagina di errore. Si può utilizzare l'oggetto eccezione solo se l'attributo è true.
- `contentType="mimeType [;charset=charSet ]" | "text/html;charset=ISO-8859-1"`: indica il tipo MIME e il codice di caratteri usato nella risposta.

Domanda 19

Come funziona la direttiva `include`

Serve ad includere il contenuto del file specificato. È possibile nidificare un numero qualsiasi di inclusioni. L'inclusione viene fatta a tempo di compilazione: eventuali modifiche al file incluso non determinano una ricompilazione della pagina che lo include.
Es:`<%@ include file="/shared/copyright.html"%>`

Domanda 20

Differenza scope page e scope request

- **Scope page**: visibilità solo nella pagina corrente, fino a quando la pagina viene completata o fino al forward.
- **Scope request**: visibilità in tutte le pagine incluse, cioè la pagina corrente ed eventuali altre pagine incluse con la direttiva `include` o `forward`.

Domanda 21

All'interno di una servlet posso vedere un JavaBean con scope session definito in una JSP? Come posso accedervi?

Dalla servlet si può invocare `session.getAttribute(id)` con argomento l'id del JavaBean.

2.3 JavaScript

Domanda 22

Definizione di pagine web dinamiche e pagine web attive e differenze

- **Pagina web dinamica**: il contenuto della pagina viene costruito ogni volta interamente dal server, in base ai parametri in ingresso e lo stato dell'applicazione, consentendo un'interattività con l'utente.
- **Pagina web attiva**: il contenuto della pagina varia lato cliente senza intervento del server d'origine.

Per rendere una pagina Web attiva si utilizza JavaScript che consente di:

- Accedere e modifcare elementi del DOM.
- Reagire ad eventi generati dall'utente.
- Validare i dati inseriti dall'utente.
- Interagire con il browser (determinare il browser utilizzato, la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc.).

Domanda 23

Come si utilizzano i cookie in JavaScript?

In JavaScript i cookie si possono ottenere con `document.cookie` che è una stringa contenente i cookie in formato `chiave=valore` separati da `;`.

2.4 AJAX

Domanda 24

Come funziona AJAX?

AJAX nasce per superare il modello sincrono classico di HTTP utilizzando un meccanismo di richieste asincrone non bloccanti al server. AJAX è composto da 3 parti:

1. Creazione dell'oggetto `XMLHttpRequest`.
2. Definizione della funzione di callback su `xhr.onreadystatechange`.
3. `open()` e `send()` della richiesta al server.

Il codice JavaScript continua quindi ad eseguire e non si blocca in attessa della risposta, ma reagisce solo all'arrivo della risposta, evitando di bloccare il browser.

Domanda 25

Quali sono gli svantaggi di AJAX?

Con AJAX troviamo criticità sia per l'utente che per lo sviluppatore come ad esempio:

- La 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.

Di solito quindi si agisce per limitare i comportamenti impropri a livello utente:

- Rendendo visibile l'andamento della chiamata.
- Interrompendo richieste che non terminano in tempo utile (tramite l'utilizzo del metodo `abort()`).

Domanda 26

Qual è la differenza tra due `send()` su un unico oggetto XMLHttpRequest e due `send()` su due oggetti separati?

- **Due `send()` su un unico oggetto**: si utilizza meno memoria ma nel caso di necessità della risposta si è forzati ad **aspettare entrambi** i risultati.
- **Due `send()` su due oggetti separati**: si consuma leggermente più memoria ma si ha la flessibilità di **due callback differenti** nel caso uno possa richiedere tempo o dia errore.

Domanda 27

Da dove deriva JSON?

JSON è un formato per lo scambio di dati nato per sostituire XML nello scambio di dati fra Server e Client. Le sue principali caratteristiche sono:

- **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.

3 React.js

Domanda 28

Che cosa sono i *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.

Domanda 29

Che cos'è JSX e come funziona?

JSX (JavaScriptXml) 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()`.
JSX non è nativamente supportato dal browser (che conosce solo JavaScript), quindi **serve un compilatore** che sia in grado di trasformare il codice JSX in codice JavaScript, come ad esempio *Babel*.

Domanda 30

Qual è la differenza fra `props` e `state`?

- `props`: assumono valori **immutabili** per i quali non è prevista alcuna alterazione, utili ad esempio per configurare il componente.
- `state`: assumono valori **NON immutabili** ed è pensato proprio per contenere proprietà che nel tempo cambieranno (in seguito al verificarsi di determinati eventi).
  Quando una delle proprietà all'interno di `state` cambia valore, viene invocata la ri-renderizzazione del relativo componente(chiamata a funzione `render()`).

Domanda 31

Che cosa *triggera* la modifica della visualizzazione?

Viene triggerata dall'invocazione del metodo `setState()`.

Domanda 32

Qual è la differenza fra componenti `function` e componenti `class`?

La principale differenza fra i 2 è il fatto che i componenti `function` sono **stateless** mentre i componenti `class` sono **stateful**.

- `function`: è una semplice funzione JavaScript che prende come argomento i props e ritorna un React Element(JSX) e **NON** ha stato.
- `class`: estende `React.Component` e ha bisogno della funzione `render()` che deve ritornare un React Element, mantiene lo stato (inizializzato nel costruttore) e lo aggiorna con il metodo `setState()` (che a sua volta richiama il ri-rendering di tutti i componenti figli che dipendono da quello stato).

Domanda 33

Come funziona il Virtual DOM?

Al verificarsi di un evento, invece di manipolare il DOM del browser (come fanno JavaScript e altre librerie), **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.

Domanda 34

Come funziona il *diffing*?

All'arrivo di un evento che manipola il Virtual DOM, React.js confronta il virtual DOM appena manipolato con una copia dello stesso Virtual DOM fatta precedentemente alla manipolazione (operazione chiamata **diffing**).
Così facendo, individua solo gli oggetti che sono realmente cambiati. React.js comunica questa informazione al DOM del browser, il quale provvederà al rendering del solo oggetto modificato (Vedi esempio dei 10 checkbox).

4 Componenti

4.1 EJB

Domanda 35

Qual è la differenza fra Java Model 1 e Java Model 2?

- **Model 1**: è un pattern semplice in cui il codice responsabile per **presentazione contenuti è mescolato con logica di business**, suggerito solo per piccole applicazioni (sta diventando obsoleto nella pratica industriale).
- **Model 2**: come 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

Domanda 36

Cosa vuol dire MVC?

MVC è un pattern di programmazione (anche detto Java Model 2) formato da 3 parti:

- **Model**: rappresenta il livello dei dati, incluse operazioni per accesso e modifica. Model deve notificare View associate quando il 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 i dati debbano essere presentati.
  - Aggiorna presentazione dati quando modello cambia.
  - Reindirizza 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).

Domanda 37

Qual è la differenza fra EJB Session bean stateful e stateless?

- **Stateless**: esegue una richiesta e restituisce risultato senza salvare alcuna informazione di stato relativa al cliente; transienti, elemento temporaneo di business logic necessario per uno specifico cliente per un intervallo di tempo limitato.
  Ciclo di vita: No state -> Pooled State -> Ready State.
- **Stateful**: può mantenere stato specifico per un cliente.

Domanda 38

Qual è la differenza fra Session bean e Entity bean?

- **Session bean**
  - Lavorano tipicamente per un **singolo cliente**.
  - **Non sono persistenti** (vita media relativamente breve), **persi in caso di crash** di EJB Server.
  - **Non rappresentano dati in un DB**, anche se possono accedere e modificare questi dati.
- **Entity bean**
  - Accesso condiviso per **clienti differenti**.
  - 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**.
  - Forniscono una **vista ad oggetti dei dati** mantenuti in un database.
  - Nella maggior parte dei casi, componenti sincronizzati con relativi database relazionali.

Domanda 39

Come viene gestita la concorrenza all'interno di EJB?

La concorrenza all'interno di EJB viene gestita in due modi:

- **Resource Pooling**
  - Stateless Session Bean.
  - Entity Bean.
- **Activation/Passivation**:
  - Stateful Session Bean.

Domanda 40

Che cos'è il Resource Pooling?

Metodo per gestire la **concorrenza dei Session bean stateless**(non sarebbe possibile con i Session bean stateful dato che l'istanza EJB contiene i dati del singolo cliente e non può quindi essere condivisa con altri clienti).
L'idea base è di evitare di mantenere un'istanza separata di ogni EJB per ogni cliente, quindi l'EJB container mantiene un insieme di istanze di bean pronte per servire richieste cliente. Non esiste stato di sessione da mantenere fra richieste successive; ogni invocazione di metodo è indipendente dalle precedenti.

Domanda 41

Activation e Passivation

Activation e Passivation sono i modi con i quali l'EJB Container gestisce la concorrenza dei Session 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.

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.
La procedura è molto simile a quello che accade con la **memoria virtuale all'interno di un SO**.

Domanda 42

Ciclo di vita di uno stateless Session bean

1. **No state**: non istanziato; stato iniziale e terminale del ciclo di vita.
2. **Pooled state**: istanziato ma non ancora associato ad alcuna richiesta cliente.
3. **Ready state**: già associato con una richiesta EJB e pronto a rispondere ad una invocazione di metodo.

Una volta terminato il lavoro in Ready state, lo stateless Session Bean torna in No state.

4.2 Spring

Domanda 43

Dependency injection (Spring)

La Dependency Injection è un metodo di gestione della configurazione dei componenti e applica i principi di **Inversion-of-Control eliminando la 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).
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, dove è il componente che deve determinare l'implementazione della risorsa desiderata (**pull**).

Domanda 44

Differenza Container pesante e Container leggero

La differenza tra container leggero e container pesante è il livello di indirettezza col quale il client accede ai servizi offerti dai Beans contenuti nel Server.

- **Container pesante**: delega al container funzionalità di base come la gestione del ciclo di vita dei componenti, l'instance pooling e la Dependency Injection. Questo è reso possibile dal fatto che ogni chiamata è intercettata dal container, e vengono eseguite delle operazioni in modo trasparente al codice del cliente.
- **Container leggero**: contiene solo poche funzionalità, come ad esempio la Dependency Injection e il cliente prima ottiene il Bean attraverso la BeanFactory(che si occupa di fare la Dependency Injection) e poi invoca i metodi in modo diretto sul Bean.

Domanda 45

Cosa vuol dire MVC e come si usa in spring, ci sono controller in spring? e come si fa un controller

Spring offre supporto a componenti **"controller", responsabili per interpretare richiesta utente e interagire con business object** applicazione.
Una volta che il **controller** ha ottenuto i risultati (parte "model"), **decide a quale "view" fare forwarding del model**; view utilizza i dati in model per creare presentazione verso l'utente:

- Chiara separazione ruoli: controller, validator, oggetti command/form/ model, DispatcherServlet, handler mapping, view resolver.
- Adattabilità e riutilizzabilità: possibilità di utilizzare qualsiasi classe per controller purché implementi interfaccia predefinita.
- Flessibilità nel trasferimento model: via un Map nome/valore, quindi possibilità di integrazione con svariate tecnologie per view.
- Facilità di configurazione: grazie al meccanismo standard di dependency injection di Spring.
- Handler mapping e view resolution configurabili.
- Potente libreria di JSP tag (Spring tag library): supporto a varie possibilità di temi (theme).
- Componenti con ciclo di vita scoped e associati automaticamente a HTTPrequest o HTTPsession (grazie a WebApplicationContext di Spring).

Domanda 46

Quando si usa il tag `<ref>` su Spring?

Viene usato quando è necessario fare **injection di un bean all'interno di un altro** (target bean). Non viene fatto un controllo stretto sul tipo del bean "iniettato" rispetto a quanto definito nel target.

Domanda 47

Come funziona la `DispatcherServlet` su Spring?

Essa è vista come "Front Controller" (*vedi esempio francese colonizzatore*) e funziona in questo modo:

1. Intercetta le HTTP Request in ingresso che giungono al Web container.
2. Cerca un controller che sappia gestire la richiesta.
3. Invoca il controller ricevendo un model (output business logic) e una view (come visualizzare output).
4. Cerca un View Resolver opportuno tramite cui scegliere View e creare HTTP Response.

4.3 JSF

Domanda 48

Qual è la differenza fra JSF e JSP?

- **JSP**: documenti con parti statiche in HTML e parti dinamiche in Java.
- **JSF**: documenti XHTML con framework MVC che descrive la UI tramite componenti riutilizzabili.

Domanda 49

Templating JSF

Facilità di estensione e riuso come caratteristica generale di JSF. Templating: utilizzo di pagine come base (o template) per altre pagine, anche mantenendo look&feel uniforme.
Per costruire un template si utilizzano i seguenti tag:

- ui:**insert** – parte di un template in cui potrà essere inserito contenuto (tag di amplissimo utilizzo).
- ui:**component** – definisce un componente creato e aggiunto all'albero dei componenti.
- ui:**define** – definisce contenuto con cui pagina "riempie" template (vedi insert).
- ui:**include** – incapsula e riutilizza contenuto per pagine multiple.
- ui:**param** – per passare parametri a file incluso.

Domanda 50

Managed Bean in JSF

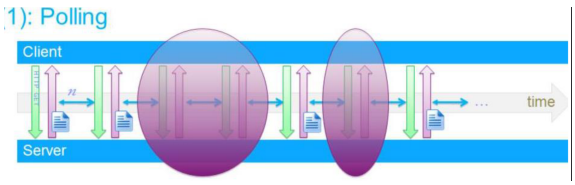
Sono dei Bean utilizzabili all'interno del Container JSF da parte di tutte le pagine che conoscano come riferirlo. Per definire un Managed Bean basta utilizzare l'annotazione `@ManagedBean`. In generale contiene logica di business (controller) il cui risultato finale è, in modo diretto o tramite invocazione di altri componenti, di produrre dati model.

5 WebSocket

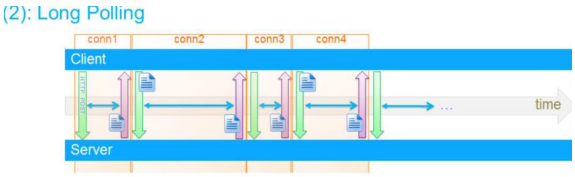
Domanda 51

Quali metodi venivano utilizzati prima delle WebSocket?

- **Polling**: Cliente fa polling a intervalli prefissati e server risponde immediatamente.
  - *Pro*: è una soluzione ragionevole quando la periodicità delle richieste è nota e costante.
  - *Contro*: è inefficiente quando il server NON ha dati da trasferire.

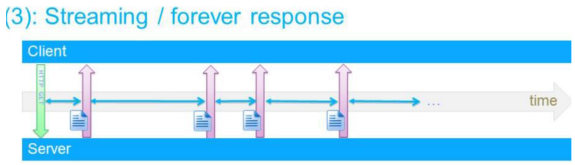


- **Long Polling**: Client manda richiesta iniziale e server attende fino a che ha dati da inviare.
  - *Pro*: quando il cliente riceve risposta, reagisce mandando immediatamente nuova richiesta.
  - *Contro*: ogni request/response si appoggia su una nuova connessione TCP.



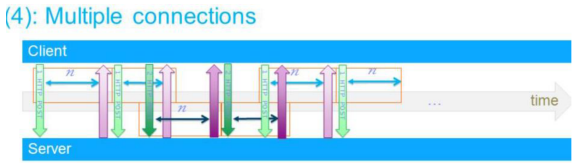
- **Streaming/Response Forever**: Client manda la richiesta iniziale e server attende fino a che ha dati da inviare.

  - *Pro*: Server risponde con streaming su una connessione mantenuta sempre aperta per aggiornamenti push (risposte parziali).
  - *Contro*: Half-duplex: solo server-to-client. Proxy intermedi potrebbero essere in difficoltà con risposte parziali.



- **Multiple Connections**: Long polling su due connessioni HTTP separate:
  1. Long Polling "tradizionale".
  2. Dati da cliente verso servitore.

  - *Pro*: Complesso coordinamento e gestione connessioni.
  - *Contro*: Overhead di due connessioni per ogni cliente e complessità di gestione.



Domanda 52

Quali sono le caratteristiche principali delle WebSocket?

- **Bi-direzionali**: Client e server possono scambiarsi messaggi quando desiderano.
- **Full-duplex**: Nessun requisito di interazione solo come coppia request/response e di ordinamento messaggi.
- **Unica connessione long running**: Unica connessione TCP per lo scambio dei messaggi.
- **Visto come un "upgrade" di HTTP**: Nessun sfruttamento di protocollo completamente nuovo, nessun bisogno di nuova "infrastruttura".
- **Uso efficiente di banda e CPU**: Messaggi possono essere del tutto dedicati a dati applicativi.

Domanda 53

Come si utilizzano le WebSocket lato Client?

```
var socket = new WebSocket("ws://server.org/ wsendpoint");
socket.onmessage = onMessage;
function onMessage(event) {
    var data = JSON.parse(event.data);
    if (data.action === "addMessage") {
        // actual message processing
    } if (data.action === "removeYessage") {
        // actual message processing
    }
}
```

Domanda 54

Perché le websocket non sono un protocollo ma un upgrade?

Perchè per l'apertura della connessione si usa HTTP con Header `Upgrade:websocket` e quindi non abbiamo bisogno di un protocollo nuovo (dato che sarebbe troppo dispensioso creare un nuovo protocollo di comunicazione).

Domanda 55

Quali sono gli eventi lato server relativi alle websocket?

```java
@ServerEndpoint("/actions")
public class WebSocketServer {
    @OnOpen
    public void open(Session session) {}

    @OnClose
    public void close(Session session) {}

    @OnError public void onError(Throwable error) {}
    I
    @OnMessage public void handleMessage(String message, Session session) {
        //actual message processing
    }
}
```

6 Node.js

Domanda 56

Quali sono le caratteristiche principali di Node.js?

- **Soluzione Server-Side** con codice JS compilato.
- Non ha thread o processi dedicati (**SingleThreaded**).
- Sempre **non bloccante**.
- Fortemente basato sugli eventi (**Event Loop**).
- Altamente scalabile.

Domanda 57

Come funziona l'Event Loop in Node.js?

L'Event Loop continua a leggere dalla Event Queue (facendo una `pop()`) ed esegue la funzione di callback associata a quell'evento. Ogni volta che viene completata un azione da parte del SO viene aggiunto l'evento corrispondende nella Event Queue (facendo una `push()`).
In questo modo Node.js và a ridurre fortemente overhead di context switching dato che esegue come singolo thread e reagisce solamente all'arrivo di eventi (non è mai bloccante, nemmeno per operazioni di I/O).

Domanda 58

Qual è la differenza fra Thread e Event-Driven?

- **Thread**: Basato sulla presenza di molti thread e chiamate bloccanti. Usa Context Switch.
- **Event-Driven**: Un solo thread e chiamate NON bloccanti grazie all'utilizzo di callback. NON usa Context Switch.

Domanda 59

Cosa sono i moduli Node?

Il core di Node consiste di circa una ventina di moduli, alcuni di più basso livello come per la gestione di eventi e stream, altri di più alto livello come HTTP. ll core di Node è stato progettato per essere piccolo e snello; i moduli che fanno parte del core si focalizzano su protocolli e formati di uso comune.

Domanda 60

Esempio di librearia asincrona in Node

```
1   var fs = require("fs");
2
3   fs.readFile('inputfile.txt', readDoneCallback);
4
5   function readDoneCallback(error, data) {
6       if (error)
7           return console.error(error);
8       console.log(data.toString());
9       console.log("End of Program execution");
10  }
```