Capitolo 1 - Introduzione

domenica 17 aprile 2022 11:54

1. INTRODUZIONE

1.1. World Wide Web (WWW):

- Il padre è stato Tim Berners-Lee
- 1989-1991
- L'idea su cui si basa è l'IPERTESTO
- Componenti chiave sono: versione iniziale di HTML, versione iniziale di HTTP, server web e browser
- L'applicazione browser è un client HTTP che prepara le richieste HTTP da inoltrare al server web che le interpreta per poi dare le risposte.
- La risposta HTTP è un documento HTML da visualizzare.
- Una delle funzionalità di un browser è quella di visualizzare oggetti web (documenti HTML e non solo)

1.1.1. Ipertesti:

- Premessa: un documento testuale o testo ha una struttura sequenziale->stringa di caratteri che viene letta dall'inizio alla fine.
- L'ipertesto è un testo a cui sono stati aggiunti degli archi con direzione o link che rappresentano le associazioni->viene navigato->la direzione indica il verso di lettura.
- I link possono essere interni (verso una parte del documento) oppure esterni (ad un diverso documento)
- Struttura a grafo che viene ereditata anche dal web->grande ipertesto in cui i link esterni vengono seguiti attraverso richieste HTTP.
- I nodi possono riferirsi anche ad oggetti web(immagini, suoni, filmati etc)
- SGML: linguaggio che permette di definire una grammatica per testi, in particolare mark-up. HTML è un'istanza di SGML
- Altri linguaggi utili: XML e XHTML
- HTML: linguaggio di etichettatura per gli ipertesti. Un browser deve, dunque, permettere di seguire i link ovvero preparare una richiesta HTTP da spedire al server web identificato dal link per ottenere la risorsa

1.1.4. Cosa si intende per World Wide Web?:

- Applicazione client/server operante su Internet e reti intranet TCP/IP
- Opera attraverso protocollo HTTP

2. SERVIZI FORNITI DA INTERNET

1.2.1. Terminali, client e server

- Computer collegati a Internet sono detti host o terminali o end system
- Un'applicazione client usa un servizio da un programma server->client e server possono essere eseguiti sulla stessa macchina oppure su macchine diverse->solitamente si svolgono su macchine diverse
- Struttura client/server->prevalente per le applicazioni Internet->applicazioni distribuite
- Non tutte le applicazioni di un client interagiscono con un server->per la condivisione di file peer-to-peer, l'applicazione agisce da client ma in alcuni casi anche da server

1.2.3. Protocolli:

- Tutte le attività in Internet->gestite da un protocollo che: definisce il formato e ordine dei messaggi scambiati tra due o più entità comunicanti; definisce le azioni che hanno luogo a seguito della trasmissione e/o ricezione di un messaggio o di altri eventi
- Protocolli più importanti: TCP (controlla la trasmissione), IP (specifica il formato dei pacchetti che sono scambiati tra router e terminali), HTTP (i browser che implementano il lato client dell'HTTP. Molto spesso sono i programmi a fungere da client ed è per questo che quando ci si riferisce al client di un'applicazione web si usa il termine user agent)
- Il server web funge da deposito di oggetti web->i quali sono indirizzabili da un URI e vanno ad implementare il lato server HTTP

1.2.4. Pagine web:

- Insieme di oggetti web il cui rendering viene fatto contemporaneamente perché strettamente collegato (pagina con testo ed immagini)
- Oggetto web->risorsa->file indirizzabile da un URL come un file HTML, un JPEG o GIF, un applet JAVA etc
- URL ha la forma :dettagli
- Differenza tra URN e URL: il primo serve slo a identificare univocamente un oggetto mentre l'URL ci dice come arrivare a una copia di quella risorsa->danno informazioni complementari.
- URL: informazioni sul nome del protocollo, sul nome dell'host su cui si trovano gli oggetti e sul path degli oggetti sull'host
- Browser è un tipo di user agenti->interfaccia tra utente e applicazione->come lo sono anche i sistemi di Information Retrieval che si muovono nel web alla ricerca di informazioni e gli interpreti VXML.
- La connessione riguarda un solo server anche se due interazioni successive possono riguardare due server anche lontanissimi tra loro

3. DAL WEB STATICO AL WEB DINAMICO

- Premessa: il web era monodirezionale ovvero il client inviava una richiesta ad un server che gli inviava un file, visualizzato dal browser sul client->limitazioni
- La comunicazione è diventata bidirezionale->il client può passare informazioni al server->parametrizzare la richiesta oppure a passare al server in modo definitivo alcuni dati->tutto viene supportato dal protocollo HTTP->programmazione lato server che serve a leggere ed elaborare i parametri
- L'informazione deve essere mostrata nello stesso modo da tutti i client e quindi da tutti i server->portabilità inter-browser
- Non esiste uno standard a cui i browser debbano confrontarsi
- I primi browser erano molto statici->l'esecuzione di qualsiasi compito era data al server ->inefficiente ma sicuro
- Le dinamicità che sono state introdotte: fogli di stile (definire la rappresentazione del documento adattandola ai vari media disponibili lato client) e la programmazione lato client

- Primo meccanismo per raccogliere ed elaborare le informazioni lato server è il CGI-> cosa fare con la sottomissione si trova codificata nella richiesta HTTP stessa: una delle azioni più comuni è di eseguire un programma che sta sul server nella directory (cgi-bin)->scritti in qualsiasi linguaggio->problemi collegati sono: manutenibilità e il tempo di risposta (dimensione dei dati trasferiti, carico del server, carico della rete).
- Programmazione lato client: il browser esegue parte del lavoro rendendo possibile una maggiore interattività ed efficienza->maggiore sforzo computazionale al client->programmazione all'interno del browser->plug-in grazie ai quali vengono aggiunte nuove funzionalità al browser scaricando un pezzo di codice che si incastra nel browser così da poter svolgere l'attività->si scarica un plug-in alla volta->funzionalità veloci e potenti
- Plug-in->implementazione non banale->linguaggi di scripting per la programmazione lato client->inserire codice direttamente nella pagine HTML->attivato dal browser durante la visualizzazione della pagina. Vantaggi: facile da usare e da comprendere, veloci da caricare. Svantaggio: codice esposto e facile da copiare. Nel browser web vengono usati per risolvere problemi specifici->creazione di GUI
- Linguaggi di scripting più diffusi: JavaScript, VBScript e TCL/TK ma si usano anche gli Applets Java
- Applet: programma che viene eseguito nel browser->scaricata come un qualsiasi oggetto web all'interno della pagina e quando attivata manda in esecuzione il programma->software distribuito dal server al client quando ce n'è bisogno->viene scaricata in bytecode.

1.3.1. Programmazione lato server

- Richiesta che riguarda degli oggetti web
- Viene, anche, coinvolta una base di dati: richiesta già stata elaborata dal lato client, viene comunque inviata al server che la elabora e inoltra alla base di dati e prepara la risposta della base di dati in una pagina web
- Vengono inclusi anche servlets e portlets

4. INTERNET VS INTRANET

- Applicazioni sviluppate sul web molto numerose->porting su reti intranet aziendali
- Attenzione sul codice sviluppato per il lato client->eseguito su piattaforme molto diverse->per l'intranet si usa un'ambiente molto più controllato, noto a priori->codice legacy
- Aggiornamenti codice fatti tramite browser

5. SICUREZZA WWW

- Rischio di diffusione di virus
- File di vari tipi: file GIF, codice di script, codice Java compilato, componenti Active X
- Firme digitali->certificano chi è l'autore, tempo trascorso da quando il codice viene scaricato a quando viene rilevato il danno
- Web vulnerabile anche agli attacchi contro i server
- Attacchi possono essere: passivi (intercettazione del traffico di rete tra un browser e un server oppure accesso a informazioni riservate), attivi (simulazione di altri utenti, modifica dei messaggi in transito fra client e server e l'alterazione delle informazioni contenute in un sito web: server web, browser web e traffico di rete in transito tra server e browser

Capitolo 2- Protocollo HTTP

domenica 17 aprile 2022 13:24

2. IL PROTOCOLLO HTTP

2.1. Introduzione:

- È un protocollo senza stato->non prevede né che il server né che il client mantengano informazioni sullo stato della comunicazione
- Se un'applicazione web deve mantenere informazioni sullo stato della comunicazione->implementare meccanismi
- HTTP deve definire: tipo dei messaggi scambiati, sintassi dei vari tipi di messaggio, significato dell'informazione in essi contenuta e le regole che determinano quando e come un processo invia messaggi o risponde a messaggi
- HTTP prevede solo due tipi di messaggi in forma testuale: richiesta (client->server) e risposta (server->client)
- Esempio richiesta: GET/somedir/page.html HTTP/1.1
- I messaggi sono formati da: request line nella richiesta (prima riga) e status line nella risposta, righe di intestazione (varie o nessuna), riga vuota, body del messaggio
- Request line->3 campi:

a. **METODO:**
   i. HTTP/1.0: GET (richiede l'oggetto), POST (invia l'informazione) e HEAD (restituisce solo l'intestazione->debugging)
   ii. HTTP/1.1: PUT (carica oggetto), DELETE (cancella oggetto) e altri come TRACE, CONNECT etc
b. **URL:** oggetto richiesto
c. **VERSIONE HTTP**
d. **LINEE DI INTESTAZIONE**
   i. Host
   ii. Connection: close-> non usa una connessione persistente
   iii. User-agent: permette di condizionare la risposta sul tipo di user-agent
e. **BODY**

- Nello scambio dell'informazione bidirezionale, l'informazione passa dal client al server all'interno della richiesta HTTP in un modo che dipende dal metodo HTTP adottato->se uso GET allora l'informazione viene passata come parametro alla fine dell'URL se uso POST viene passata nel body del messaggio
- Esempio risposta:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html

dati ... dati ...
```

- Prima riga->status line ed ha 3 campi:

a. **VERSIONE DEL PROTOCOLLO**
b. **CODICE DI STATO:**
   i. 200 OK
   ii. 301 MOVE PERMANENTLY-> il nuovo URL sta nell'intestazione in Location:
   iii. 400 BAD REQUEST
   iv. 404 NOT FOUND
   v. 505 HTTP VERSION NOT SUPPORTED
c. **Corrispondente messaggio di stato**

- Codici:

a. **1xx**: informativo, la richiesta è stata ricevuta e l'elaborazione continua
b. **2xx:** successo, la richiesta è stata ricevuta ma anche compresa ed accettata
c. **3xx:** redirezione, sono necessarie ulteriori operazioni per il buon fine della richiesta
d. **4xx:** errore lato client, la richiesta contiene errori sintattici per questo non viene accettata
e. **5xx:**errore lato server, richiesta corretta ma il server non riesce a soddisfarla

- Il file corrispondente all'oggetto viene trasferito così com'è->responsabilità del browser o dell'UA sapere come visualizzarlo (elaborarlo) e non è detto che il rendering implementata da due client diversi sia la stessa.
- HTTP SOLO client->server web
- Informazione sul tipo di file, server->client: riga d'intestazione Content-Type: ma se è assente si usa text/html
- **ATTENZIONE:** con HTTP/1.1 occorre mettere una riga di header con Host anche se è vuota

2.2.1.Meccanismo di cache:

- Un sito può anche adottare un meccanismo di cache->invece di scaricare tutta la pagina ogni volta che accediamo al sito ne conserveremo sempre l'ultima copia e la scarichiamo quando è necessaria
- Occorre che il client sia in grado di controllare se la pagina richiesta dall'utente è stata modificata oppure se quella che è stata salvata in cache è ancora valida senza riscaricare la pagina->IF-MODIFIED-SINCE da inserire nell'intestazione della richiesta e il codice 304 NOT MODIFIED da inserire nella status line della risposta

2.2.2. Session tracking: cookies e autenticazione:

- L'adozione del session tracking è reso necessario dal fatto che HTPP è stateless-> i messaggi non contengono informazioni sugli scambi precedenti-

>sono informazioni necessarie in alcuni casi per lo svolgimento delle funzionalità richieste dall'utente->programmazione lato server e/o lato client

- Due strategie di session tracking

a. **Session tracking via cookies:** un cookie è un pezzo di testo che viene scambiato tra client e server secondo opportune modalità. Contiene: host e path dell'applicazione, exipiration date, nome e valore del cookie e altro. Ogni cookie viene creato dal server che lo passa al client nel campo set-cookie dell'intestazione della risposta HTTP->il client decide se salvarlo. Ogni volta che lo UA prepara una richiesta per una certa applicazione identificata all'interno dell'host da un particolare path, inserisce all'interno dell'intestazione della richiesta a tutti i cookie che ha memorizzato che corrispondono a quella combinazione di host e path. Quindi se ripeto la stessa richiesta, il browser aggiungerà all'header una riga col cookie->il server riceve il cookie e quindi evita di inserire nella risposta il campo Set-cookie->programmazione lato server.

i. **EXPIRATION DATE:** serve a consigliare allo user-agent quando cancellare il cookie->se è negativo allora lo si dovrebbe cancellare immediatamente perché ogni cookie è determinato da host, path e name in maniera univoca->se il server manda un cookie con la stessa combinazione allora non viene sovrascritto. Se il server vuole che un cookie venga cancellato allora ne invia uno con lo stesso nome di quello da cancellare, in modo che quest'ultimo venga prima sovrascritto e immediatamente cancellato.

b. **Session tracking via autenticazione:** se proviamo ad accedere ad un sito per una demo per l'autenticazione basata su HTTP avremo il 401 AUTHORIZATION REQUIRED-> UA deve procurarsi le credenziali per accedere al sito->ad esempio username e password->non solo sono aggiunte nella richiesta che sottopone al server ma le salva anche per la richiesta successiva

- L'HTTP fornisce anche un meccanismo che permette al client di impiegare il caching per assicurando che gli oggetti passati al browser siano aggiornati->GET condizionato

Capitolo 3- HTML

mercoledì 20 aprile 2022 12:40

3. INTRODUZIONE

- HTML è un linguaggio di etichettatura->indicazioni su come il testo andrà formattato
- Indicare i diversi paragrafi, livelli di titolo, liste numerate etc
- **Esempio:**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Qui va il titolo del documento</title>
</head>
<body>
qui va il contenuto del documento
</body>
</html>
```

- Prima riga->quale versione di HTML si adotta
- Documento HTML composto da un insieme di elementi->elemento=parte di documento compreso tra l'apertura di un'etichetta(<head>) e la relativa chiusura(</head>), etichette comprese. L'etichetta (tag) rappresenta il nome dell'elemento
- **ATTENZIONE:** non bisogna confondere il concetto di etichetta con quello di elemento. L'elemento è una parte di documento mentre l'etichetta è il nome che caratterizza il tipo dell'elemento. In un documento possono esserci più elementi con la stessa etichetta
- HTML non dà indicazioni precise sull'aspetto grafico ma solo sulla funzione delle diverse parti di testo->si introducono i fogli di stile per maggiori aspetti della rappresentazione grafica->in particolare il CSS
- **CSS:** descrive il layout grafico o sonoro del documento stesso->rendering
- HTML prevede l'introduzione dei link

```
<p>This a link to <a href="peter.html">Peter's page</a>.</p>
<p><a href="../mary.html">Mary's page</a></p>
<a href="friends/sue.html">Sue's page</a>
<a href="../college/friends/john.html">John's page</a>
This is a link to <a href="http://www.w3.org/">W3C</a>.
<a href="/"><img src="logo.gif" alt="home page"></a>
<h2 id="night-spots">Local Night Spots</h2>
<ul>
  ...
  <li><a href="#night-spots">Local Night Spots</a></li>
  ...
</ul>
```

- / indica la radice dell'albero delle directory visibili da web->homepage->etichette anche per l'inclusione di immagini, suoni e video
- Etichette per interagire con l'utente->l'utente può dare origine a degli eventi a cui verranno associate diverse azioni->moduli o form per la raccolta dati e di bottoni per la creazione di una richiesta HTTP da inoltrare->le form non sono L'UNICO metodo per passare parametri al server
- La presenza di un form implica programmazione lato server per implementare un programma che prenda in ingresso i parametri raccolti dalla form e li elabori in modo opportuno.

3.1.1. Differenza tra gli attributi name e ID

- **Name:** nome del parametro che viene passato al server->anche più parametri con lo stesso nome
- **ID:** univoco->al di fuori delle form conviene sempre usare id al posto di name

3.2. XML E XHTML

- **XML:** linguaggio di markup estendibile->etichette possono essere scelte dall'utente->descrivere i dati
- **XHTML:** variante di HTML che segue la sintassi di XML->stesse etichette di HTML->il documento è anche XML

3.3 HTML5:

- HTML4 aveva già introdotto specifiche importanti come la definizione di un unico linguaggio HTML, definisce dettaliati modelli di processo per favorire la creazione di applicazioni interoperabili, migliora il markup per i documenti, introduce nuove etichette e API peer le varianti dell'HTML
- Le novità introdotte dall'HTML5 rispetto all'HTML4 sono finalizzate soprattutto a migliorare il disaccoppiamento fra struttura, definita dal markup, caratteristiche di resa (tipo di carattere, colori, eccetera), definite dalle direttive di stile, e contenuti di una pagina web, definiti dal testo vero e proprio. Inoltre l'HTML5 prevede il supporto per la memorizzazione locale di grandi quantità di dati scaricati dal web browser, per consentire l'utilizzo di applicazioni basate su web.

3.3.1. Compatibilità con le precedenti versioni:

- Vengono introdotti i requisiti che si dividono in: per gli autori e per gli user agent

3.3.2. Motivazioni:

- Lo sviluppo di pagine di contenuti da pubblicare in web ormai non può prescindere da un gruppo di strumenti che si integrano strettamente l'uno con l'altro

a. **HTML** per l'annotazione degli ipertesti->separare le indicazioni di formattazione con quelle legate ai contenuti del testo->portabilità e manutenibilità
b. **CSS:** formattazione nel rendering dei contenuti
c. **DOM:** indicazioni su come il documento HTML viene rappresentato in memoria mediante API

d. **JavaScript**: permette di elaborare la DOM per eseguire operazioni lato client
- In HTML si introducono nuove features che siano basate su HTML, CSS, DOM e JavaScript
- Ridurre la necessità di plugin esterni
- Migliorare la gestione degli errori
- Sostituire l'uso di un linguaggio di scripting con adeguato markup
- Indipendenza dal device
- Il processo di sviluppo delle specifiche deve avvenire in maniera chiara

3.3.3. Doppia sintassi
- Due possibili sintassi

a. text/html
b. application/xhtml+xml oppure application/xml

3.3.4.Memorizzazione
- Memorizzare dati localmente all'interno del browser->in alternativa ai cookies (coppia di stringhe nome=valore)
- Non vengono inseriti in ogni richiesta Http inoltrata al server->solo quando è necessario
- Dati memorizzati notevole->senza pesare sulle prestazioni del sito

3.4.1. Same-origin policy:
- UA eseguono azioni suggerite loro dai contenuti che scaricano dai diversi siti la maggior parte dei quali saranno stati creati dagli sviluppatori
- Occorre che lo UA applichi delle politiche per salvaguardare le informazioni di cui è responsabile->sia proprie che altrui
- Ad uno UA viene richiesto di mandare in esecuzione script prodotti dal sito che sta visitando->contro ogni azione malevole->si usa il concetto di origine per implementare la cosiddetta same-origin policy-> i contenuti possono interagire senza restrizione con tutti i contenuti provenienti dalla stessa origine->non può avvenire in maniera diversa.
- Basa la fiducia sul suo URL
- Due URL sono considerate appartenere alla stessa origine se hanno uguali schema, host e porta

3.4.2. Mancata validazione dell'input dell'utente;cross-site scripting;SQL injection
- Quando si introducono delle form nel documento HTML che viene servito al client->molta attenzione al testo che viene effettivamente inserito e all'uso che se ne fa->la trascuratezza possono comportare non solo comportamenti inattesi da parte del sito ma anche danni al sito stesso o al sito con cui si sta interagendo->cancellazione di tutti i dati sul server
- Prevedere filtri per la validazione dei dati in ingresso->enumerazione dei casi favorevoli->se al contrario ci si limita a bloccare solo gli ingressi che si ritengono pericolosi->**blacklist-based**
- **Query string:** parte dell'URL che segue il punto di domanda e contiene i parametri e ridirige l'utente ad uno script CGI che mostra quel messaggio-> pericoloso nel caso in cui il messaggio venga presentato all'utente cosi come viene ricevuto nella query string senza introdurre i caratteri escape->**cross-site scripting attack**
- Anche gli elementi img devono essere validati sulla base di una **whitelist** anche gli attributi per evitare che un eventuale attaccante possa usare l'attributo onload per mandare in esecuzione uno script di sua scelta
- La costruzione di URL->basata su input->basato su **whitelist**
- Se si permette l'inserimento di un elemento di tipo base, tutti gli elementi script di quella pagina possono venir piratati

3.4.3. Cross-site request forgery (CSRF)
- In un sito in cui si permette all'utente di usare la sottomissione di form per portare a termine delle azioni a nome dell'utente, come inviare dei messaggi, compiere degli acquisti, richiedere documenti, è della massima importanza verificare che l'utente abbia fatto la richiesta intenzionalmente e non perché tratto in inganno.
- L'origine di questo problema risiede nel fatto che HTML permette all'utente di sottomettere le form anche a origini diverse da quelle dell'applicazione->popolare le form co token nascosti caratterizzanti l'utente oppure di controllare il campo <Origin nell'intestazione di tutte le richieste HTTP.

3.4.4. Clickjacking
- Bisogna garantire che non sia possibile che l'utente le scelga senza averne coscienza
- Si usa il iframe oppure con l'attivazione dell'interfaccia critica solo nel caso in cui si sia certi che la pagina non sia stata inclusa in un frame
- Il confronto, ad esempio, può avvenire tra l'oggetto window e il valore dell'attributo top

Capitolo 5-Java

giovedì 21 aprile 2022 11:25

5. Java per le applicazioni web

- Linguaggio orientato ad oggetti->semplice->non di scripting
- Possiede il **garbage collector**->evita l'introduzione di errori soprattutto se legati alla deallocazione
- Linguaggio fortemente tipato
- Non ammette costrutti insicuri
- Indipendente dalla piattaforma->compilato in bytecde->eseguibile su una JVM
- Ha determinate caratteristiche di sicurezza->non possono richiamare funzioni globali, ne ottenere l'accesso a risorse del sistema in modo arbitrario
- Offre controllo maggiore degli errori->legati a due aspetti:

a. **Gestione della memoria:** occorre tener traccia di tutta la memoria allocata in modo da poterla liberare
b. **Condizioni eccezionali:** errori dovuti alla mancanza del file di ingresso

- Sintassi rigida->soprattutto per la gestione dei tipi e le dichiarazioni->rivelare molti errori a compilazione->warning solo per metodi deprecati
- Supporto al **multi-thread**->implementazione di interfacce e nella gestione di richieste concorrenti
- Ampia disponibilità di librerie
- **JVM** macchina astratta->conosce solo il formato class->le istruzioni sono contenute in un file class
- **Piattaforma Java:** linguaggio java, JVM, API libraries
- La sicurezza dipende da: il progetto del linguaggio come sicuro e di facile uso, verifica del bytecode prima dell'esecuzione, durante l'esecuzione (le classi vengono caricate e ne viene fatto il linking, opzionale ed esecuzione del programma)->il class loader definisce un namespace locale in modo che codice non sicuro non possa interferire con l'esecuzione di altri programma Java
- Mediazione della JVM->classe security manager per l'accesso alle risorse di sistema

5.1.1. Caratteristiche di sicurezza a livello del linguaggio

- In java non è possibile accedere alla memoria in modo non controllato
- È esclusa l'aritmetica dei puntatori
- Accessi non protetti alla memoria rendono possibili alcuni degli attacchi più frequenti alla sicurezza delle applicazioni
- I puntatori in Java sono le references->variabili non inizializzate:

a. Sullo **heap** vengono automaticamente inizializzate->possono avere valori non definiti
b. Sullo **stack** non vengono automaticamente inizializzate e quindi tale operazione deve avvenire in modo esplicito prima dell'uso->errore di compilazione

- Garbage collection automatica->liberare automaticamente la memoria che non viene più indirizzata->SOLO LA HEAP dove viene allocata tutta la memoria richiesta dinamicamente
- Si escludono le operazioni di cast non legali->evita che un blocco di memoria venga interpretato in modo diverso da quello che è->accesso illegale a zone di memoria

5.1.2. Caratteristiche di sicurezza a livello di macchina virtuale

- JVM usa la **sandbox** per mandare in esecuzione programmi Java: per verificare tutte le classi che entrano, per operare numerosi controlli a runtime che evitano che i programmi Java compiano azioni non valide e per alzare una barriera attorno all'ambiente di runtime e controllare tutti gli accessi all'esterno della sandbox
- Non tutti i controlli possono essere fatti durante la compilazione->il peggio che può accadere è una condizione di **denial of service** in cui il programma consuma tutte le risorse della CPU
- Un compilatore converte in bytecode (file .class) il file sorgente che può anche non essere codice Java->se ciò però non risulta essere sufficiente allora bisogna implementare anche dei controlli sull'ambiente di esecuzione
- **Esecuzione di una classe:** caricamento del file.class, esecuzione del metodo main e il caricamento delle classi accessorie
- Ogni volta che una classe viene caricata, il bytecode viene verificato. Non tutte le classi vengono verificate: tipicamente le classi caricate localmente vengono considerate affidabili e non verificate,mentre le classi caricate attraverso la rete verranno verificate.
- Controlli diversi (quindi a costi diversi) vengono applicati a seconda delle caratteristiche della classe da caricare.
- La verifica del bytecode caricato è compito del ClassLoader, che solo al termine della verifica crea le classi vere e proprie. Il ClassLoader non cerca mai di caricare le classi di java.* da remoto: in questo modo si evita che esse possano venir sostituite con classi modificate in cui i meccanismi di sicurezza siano stati in qualche modo manomessi. Inoltre, prepara uno spazio dei nomi (name space) diverso per ogni locazione da cui vengono scaricate le classi, in modo da evitare collisioni tra classi diverse che si chiamano con lo stesso nome.
- A classi provenienti da host diversi non è permesso di comunicare all'interno della macchina virtuale Java, in modo da evitare che programmi non necessariamente affidabili possano ottenere informazioni da programmi considerati affidabili.
- Una volta che le classi sono state caricate, la macchina virtuale compie i controlli necessari a tempo di esecuzione. Sono necessari controlli durante l'esecuzione sia per quel che riguarda i tipi nelle istruzioni di assegnamento che per quel che riguarda i limiti dei vettori.
- In Java, invece, i riferimenti agli oggetti includono informazioni complete sulla classe di cui l'oggetto è un'istanza: questo rende possibile il controllo della

compatibilità tra tipi a runtime. Se il controllo fallisce, viene sollevata un'eccezione.
- Ogni macchina virtuale Java in esecuzione ha al più un SecurityManager installato, responsabile per la gestione delle politiche di sicurezza. SecurityManager è una classe del pacchetto java.lang: se ne può quindi definire una sottoclasse e usare il metodo System.setSecurityManager() per stabilire un security manager personalizzato. Una volta che un manager è stato installato, ogni tentativo di sostituirlo genera un errore (viene lanciata l'eccezione Security Exception): quindi nessuno può variare questa funzione in modo ostile, rimpiazzandola. Attraverso le politiche di sicurezza definite dal Security Manager, è possibile allentare o restringere i vincoli sulle operazioni che possono essere eseguite da un programma Java: ogni tentativo di eseguire un'operazione vietata solleva unaSecurity Exception.

5.2. Java Applets

- Programmi scritti in linguaggio java che possono essere esguiti da un web browser->elaborazione lato client
- Collocati all'interno di pagine web->pagine con funzioni interattive->web dinamico
- Utilizzano la JVM o possono essere eseguiti utilizzando il Sun AppletViewer

5.2.1. Creazione e ciclo di vita di un applet

- Sono incapsulate in una pagina web ed eseguite sul client dell'utente che accede al sito->il browser scaricherà sul computer il codice applet e si occuperà, tramite JVM di interpretarlo e tradurlo in un'applicazione
- Migliorano l'aspetto, l'interattività e popolarità dei siti web
- Priva di main ma dotata di un proprio ciclo di vita->caricamento iniziale, esecuzione (con sospensioni e riprese) e una distruzione finale
- Al momento del caricamento->init() che serve per inizializzare immagini ed oggetti->start() che fa partire l'applet e se durante l'esecuzione il browser viene ridotto ad icona abbiamo lo stop() a cui segue di nuovo start() quando l'applet torna in primo piano->destroy() quando si esce dal browser che consente di effettuare eventuali azioni contestuali alla chiusura dell'applet
- Una volta completato l'applet, sarà necessario includerlo in una pagina web utilizzando i tag HTML per specificare la classe che implementa l'applet, le dimensioni della pagina e altri parametri opzionali.
- Una volta copiato il jar contenente l'applet nella stessa directory dove è salvata la pagina HTML->possibile che sia sprovvista di un archivio jar
- È importante notare infine che è possibile passare dalla pagina HTML dei parametri all'applet, per effettuare tale operazione è sufficiente specificare i parametri e i relativi valori all'interno della definizione di applet.

5.2.2. Firmare un applet

- Fornire maggiore flessibilità e potenza agli applet ma anche garantire rispetto della sicurezza
- La firma non è altro che l'attestazione di proprietà e di origine del sw. Così chi si ritrova a caricare tale applicazione ha la possibilità di decidere se ritenere o meno affidabile l'origine dell'applet e, solo in caso affermativo, di rimuovere le limitazioni ad essa imposte.

5.2.3. Utilizzo

- Incapsulati all'interno di pagine web, gli applet sono utilizzati per fornire contenuti interattivi che il linguaggio HTML non è in grado di offrire.
- Per eseguirne il contenuto, la maggioranza dei web browser utilizza una sandbox, in modo da impedire agli applet di accedere alle informazioni salvate in locale sul computer-> il codice sorgente degli applet viene scaricato dal web server attraverso il web browser ricevendo anche la pagina HTML che lo contiene.
- In alternativa, l'applet può provvedere ad aprire una finestra personale senza doversi appoggiare al codice HTML per mostrare l'interfaccia grafica.
- Gli applet mostrati nelle pagine web sono identificati dal tag HTML <applet>... </applet> (non ufficiale), al quale è preferibile <object></object>. Il tag HTML utilizzato specifica la posizione del sorgente dell'applet da scaricare.
- Gli applet Java possono essere eseguiti senza problemi dai browser delle principali piattaforme.

5.2.4. JNLP e Java Web Start

- Le applets hanno sempre il pesante vincolo di dover essere eseguite all'interno di un browser->con maggiore carico computazionale per il client che deve mandare in esecuzione anche il browser e con un impatto visivo non trascurabile visto che vengono visualizzati anche tutti i pannelli di comandi del browser
- JNLP permette di scaricare ed eseguire una applet anche fuori dal browser->scaricare dinamicamente risorse da Internet durante l'esecuzione->versione controllata automaticamente
- JNLP è un protocollo->ne serve un'implementazione, tipo JAWS->un'applicazione basata su JNLP prevede JAR file contenente un'applicazione standard e un file XML per indicare al client come scaricare e installare l'applicazione.

Capitolo 6-Servlet

venerdì 22 aprile 2022 10:44

6. INTRODUZIONE

- Uso di Java per la programmazione web lato server
- Maggiore efficienza, facilità di sviluppo, sicurezza, maggiore condivisione dell'informazione sia all'interno dell'applicazione che col server

6.1. Confronto tra servlet e CGI

- Adattamento nella programmazione lato server alle caratteristiche proprie delle applicazioni web->server riceve tipicamente molte richieste in contemporanea e deve gestirle senza problemi di sincronizzazione e con occhio alle prestazioni
- Primo passo->CGI
- Le servlet permettono di condividere delle informazioni sia all'interno di un'applicazione che col server mentre nelle CGI l'unico ambiente di definizione (scope) previsto è quello di una richiesta->in corrispondenza di ogni richiesta, viene avviato un processo che termina una volta che la richiesta viene soddisfatta
- Condividere un'informazione tra due richieste diverse, della stessa risorsa e/o applicazione->occorre che tale informazione venga salvata in modo persistente->la sincronizzazione tra i diversi accessi a questa informazione persistente o viene gestita dal database management system o va esplicitamente considerata
- Il CGI non dà modo ai programmi CGI di interagire direttamente col server o comunque di usare le funzionalità del server dopo che il programma è entrato in esecuzione->eseguito come processo separato
- Nella servlet->ad ogni richiesta corrisponde un nuovo thread all'interno del servlet engine->thread diversi possono accedere a variabili comuni->condividere informazioni tra richieste diverse alla stessa servlet->possono esserci problemi di sincronizzazione tra richieste diverse
- Manca alle servlet la separazione tra aspetti grafici e aspetti di programmazione
- Una servlet permette di implementare delle funzionalità eseguite lato server->quando un web server riceve una richiesta per un programma CGI esso:

a. Dà origine ad un nuovo processo per eseguire il programma
b. Passa a questo processo le informazioni necessarie per costruire la risposta via standard input e variabili d'ambiente

- La creazione di un nuovo processo per ogni richiesta ricevuta richiede risorse sia in termini di tempo che di memoria da parte del server->limita le risposte che possono essere elaborate contemporaneamente
- Alcune alternative

a. **FastCGI:** crea un singolo processo persistente per ogni programma CGI->in caso di richieste concorrenti occorre comunque duplicare i processi->non risolve il problema dell'interazione col server
b. **Perl**
c. **ASP**
d. **JavaScript lato server**
e. **PHP**

6.1.2. Come si presenta una servlet

- Classi java che possono venir caricate dinamicamente per estendere le funzionalità del server
- Eseguite nella JVM sul server->sicure e portabili
- Operano solo sul server e non richiedono alcun supporto da parte del browser
- Le servlet vengono gestite all'interno del processo del server da thread separate->due vantaggi: efficienza e scalabilità, ottima interazione col server
- Offrono un'ottima portabilità sia su sistemi operativi diversi che su server e servlet engines diversi
- Anche le **JSP** vengono tradotte in servlets e forniscono quindi un sistema per implementare il meccanismo di richiesta e risposta del server senza doversi occupare di tutti i dettagli delle servlets
- Si usano:

a. Le JSP quando la maggior parte del contenuto passato al client è testo statico e markup mentre solo una piccola porzione del contenuto viene prodotta dinamicamente mediante codice Java
b. Le servlets quando la porzione di testo da produrre staticamente è poca->non producono testo ma portano a termini i compiti richiesti dal client e alla fine richiama altre servlets o JSPSs per produrre risposta

- Le servlet passano il risultato al client sottoforma di file HTML, XHTML e XML da visualizzare nel browser ma anche in altri formati
- Le servlet possono estendere ogni tipo di server->devono implementare l'interfaccia Servlet che è la classica API di un fornitore di servizi
- Le interfacce in Java hanno analogie con le classi ma non hanno variabili di istanza e pur avendo dichiarato i metodi->non hanno alcun body associato->ogni classe può implementare un numero qualsiasi di interfacce
- Per implementare un'interfaccia->classe definisca un'implementazione per tutti i metodi dell'interfaccia in modo che i prototipi coincidano
- Metodi dell'interfaccia Servlet->invocati automaticamente dal servlet engine

public void init(ServletConfig config)

throws ServletException;

public ServletConfig getServletConfig();

public void service(ServletRequest req, ServletResponse res)

throws ServletException, IOException;

public String getServletInfo();

public void destroy();

}

- 5 metodi però non dà il body:

a. **Primo rigo:** inizializza la servlet una sola volta durante il ciclo di esecuzione del servlet
b. **Terzo rigo:** ritorna un oggetto che implementa l'interfaccia ServletConfig e fornisce l'accesso alle informazioni sulla configurazione della servlet->parametri di inizializzazione e il ServletContext
c. **Penultimo rigo:** viene definito in modo da restituire una stringa con informazioni quali autore e versione
d. **Quarto rigo:** viene mandato in esecuzione in risposta alla richiesta mandata da un client alla servlet
e. **Ultimo rigo:** invocato quando la servlet viene terminata->usato per rilasciare le risorse

- Il metodo init() viene invocato solo quando la servlet viene caricata in memoria, di solito a seguito della prima richiesta per quella servlet-> solo dopo che il metodo è stato seguito, il server può soddisfare la richiesta del client invocando il metodo service che riceve la richiesta, la elabora e prepara la risposta per il client
- il metodo service viene richiamato ogni volta che si riceve una richiesta per ogni nuova richiesta il servlet engine crea una nuova thread di esecuzione in cui viene mandato in esecuzione service. Solo quando il servlet engine termina la servlet, viene invocato il metodo destroy per rilasciare le risorse della servlet.
- I pacchetti delle servlet definiscono due classi astratte che implementano l'interfaccia Servlet:

a. **GenericServlet** (in javax.servlet);
b. **HttpServlet** (in javax.servlet.http).

- Queste classi forniscono un'implementazione di default per tutti i metodi dell'interfaccia.

public abstract class {\bf HttpServlet} extends GenericServlet implements java.io.Serializable

- Si implementa una servlet rispetto al protocollo HTTP, e quindi per il web. Essa fornisce una classe astratta che verrà estesa dall'implementazione delle classi che lo sviluppatore implementa per realizzare la propria applicazione. Ogni sottoclasse di HttpServlet deve fare overriding di almeno un metodo, di solito uno dei seguenti:

a. **doGet**, se la servlet viene richiamata col metodo HTTP GET;
b. **doPost,** per richieste HTTP con metodo POST;
c. **doPut**, per richieste HTTP con metodo PUT;
d. **doDelete,** per richieste HTTP con metodo DELETE;
e. init e destroy, per gestire eventuali risorse necessarie al funzionamento della servlet (ad esempio, connessioni con la base di dati);
f. **getServletInfo**, usata dalla servlet per fornire informazioni riguardanti se stessa.

- I metodi doxxx prendono in ingresso:

a. **Un oggetto ServletRequest,** corrispondente ad uno stream di ingresso da cui leggere la richiesta del client
b. **Un oggetto ServletResponse,** corrispondente ad uno stream di uscita su cui scrivere la risposta

- Possono essere basati su bytecode oppure caratteri
- In caso di problemi, viene lanciata una **ServletException** o una **IOException**. In caso di richieste multiple, diverse chiamate al metodo service verranno eseguite in parallelo
- Le servlet possono anche implementare l'interfaccia **javax.servlet.SingleThreadModel**
- Funzionamento basato su multithreading

6.1.3. La classe HttpServlet

- Le servlet web estendono HttpServlet
- poiché una sola servlet di quella particolare classe viene caricata nel servlet engine, e non viene mai scaricata se non quando viene terminato il servlet engine stesso, cosa che avviene di norma al reboot della macchina, gli attributi della classe divengono di fatto oggetti persistenti.
- Per non perdere l'informazione quando il servlet engine viene, per qualsiasi motivo, terminato, si utilizzano i metodi init() e destroy(), che vengono chiamati automaticamente quando la servlet viene caricata e scaricata, e in ogni caso in cui il servlet engine viene terminato.
- Il metodo service() viene sovrascritto per tener conto delle diverse richieste che arrivano ad un client Web. Il metodo service() prima di tutto decide quale richiesta è stata inoltrata, e poi chiama il metodo opportuno: ad esempio, doGet() o doPost() a seconda del tipo di richiesta.Altre chiamate di uso molto meno frequente: doDelete, doOptions, doPut, doTrace.
- In ingresso due parametri

a. **HttpServletRequest request**->rende accessibili i dati relativi alla richiesta inoltrata dal client
b. **HttpServletResponse response**->rispsta da passare al client e hanno tipo di ritorno void

6.1.4. L'interfaccia HttpServletRequest

- I metodi dichiarati provendono dall'interfaccia **ServletRequest**
- Un'implementazione dell'interfaccia è data dalla classe **HttpServletRequestWrapper**.

a. **String getParameter(String name):** torna il valore del parametro name passato nella get/post;
b. **Enumeration getParameterNames():** returns an enumeration of string objects containing the names of the parameters contained in this request. "
c. **String[] getParameterValues(String name**): se name ha valori multipli
d. **Cookie[] getCookies()**: memorizzati dal server nel client;
e. **HttpSession getSession(boolean create):** se l'oggetto non esiste e create =true, allora lo crea.

6.1.5.L'interfaccia HttpServletResponse

- **void addCookie(Cookie cookie)**: nell'header della risposta passata al client; essa viene memorizzata a seconda dell'età massima e dell'abilitazione o meno ai cookie dispostanel client;
- **ServletOutputStream getOutputStream():** stream basato su byte per dati binari;
- **PrintWriter getWriter():** stream basato su caratteri per dati testuali;
- **void setContentType(String type):** specifica il formato MIME.

```
// Una semplice servlet per elaborare una richiesta di tipo get

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class SempliceServlet extends HttpServlet {

   // elabora le richieste get dal client
   protected void doGet( HttpServletRequest request,
      HttpServletResponse response )
         throws ServletException, IOException
   {
      response.setContentType( "text/html" );
      PrintWriter out = response.getWriter();

      // send XHTML page to client
      // start XHTML document
      out.println( "<?xml version = \"1.0\"?>" );

      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
         "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
         "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );


     out.println(
        "" );

     // head section of document
     out.println( "" );
     out.println( "<h4>Un semplice esempio di servlet</h4>" );
     out.println( "" );

     // body section of document
     out.println( "" );
     out.println( "<b>Welcome to Servlets!</b>}" );
     out.println( "" );

     // end XHTML document
     out.println( "" );
     out.close(); // lo stream va chiuso
   }
}
```

Alla servlet possiamo anche passare dei parametri: ecco un esempio in cui viene passato il nome proprio della persona:

```
public class SecondaSempliceServlet extends HttpServlet {
   // ancora la richiesta di tipo get
   protected void doGet( HttpServletRequest request,
      HttpServletResponse response )
         throws ServletException, IOException
   {
      String firstName = request.getParameter( "firstname" );

      response.setContentType( "text/html" );
      PrintWriter out = response.getWriter();

      // send XHTML document to client
```



```
      // start XHTML document
      out.println( "<?xml version = \"1.0\"?>" );
      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
      out.println(
        "" );

      // head section of document
      out.println( "" );
      out.println(
        "</h4>Elaborazione di una richiesta get che include dati<h4>" );
      out.println( "" );

       // head section of document
       out.println( "" );
       out.println(
          "</h4>Elaborazione di una richiesta get che include dati<h4>" );
```

```
      out.println( "" );

      // body section of document
      out.println( "" );
      out.println( "<h4>Hello " + firstName + "," );
      out.println( "Welcome to Servlets!</h4>" );
      out.println( "" );

      // end XHTML document
      out.println( "" );
      out.close(); // chiusura dell stream
   }
}
```

I parametri vengono passati come coppie nome/valore nelle richieste get (e, essendo un get, compaiono nell'URL, e potrebbero anche essere direttamente scritti lì).

```
// elabora una richiesta di tipo post
   protected void doPost( HttpServletRequest request,
      HttpServletResponse response )
         throws ServletException, IOException
   {
      {\bf String firstName = request.getParameter( "firstname" );}
      response.setContentType( "text/html" );
      PrintWriter out = response.getWriter();

      // send XHTML page to client
      // start XHTML document
      out.println( "<?xml version = \"1.0\"?>" );
      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
         "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
         "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
      out.println(
         "" );

      // head section of document
      out.println( "" );
      out.println(
         "<h4>Elaborazione di una richiesta di tipo post con dati</h4>" );
      out.println( "" );

      // body section of document
      out.println( "" );
      out.println( "<h4>Hello " + firstName + "," );
      out.println( "Welcome to Servlets!</h4>" );
      out.println( "" );

      // end XHTML document
      out.println( "" );
      out.close(); // chiudi lo stream
   }
}
```

Al posto di `doGet()` o `doPost()` si può comunque utilizzare anche semplicemente `service()`, come nel seguente esempio:

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

public class EchoForm extends HttpServlet {
   public void service(HttpServletRequest req, HttpServletResponse res) \
                                                      throws IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      Enumeration flds = req.getParameterNames();
      if(!flds.hasMoreElements()) { // la richiesta non contiene alcun parametro:
                                    // costruisco una form per raccoglierli
         out.print("");
         out.print("<form method=\"POST\"" +
                  " action=\"EchoForm\">");
         for(int i = 0; i < 10; i++)
            out.print("{\bf Field" + i + "} " +
                     "<input type=\"text\""+" size=\"20\" name=\"Field" + i +
                     "\" value=\"Value" + i + "\">");
         out.print("<INPUT TYPE=submit name=submit"+
         " Value=\"Submit\"></form>");
      } else {
         out.print("\section{Ecco i valori che hai inserito nella form:}");
         {\bf while(flds.hasMoreElements()) {
            String field= (String)flds.nextElement();
            String value= req.getParameter(field);}
            out.print(field + " = " + value + "");
         }
      }
      out.close();
   }
}
```

6.1.7.Servlets e multithreading

- Le richieste dei client vengono soddisfatte dal servlet engine facendo uso di un pool di thread
- Per evitare collisioni basta che il metodo service() sa thread-safe->ogni accesso a risorse comuni quali files e basi di dati va protetto dal costrutto synchronized

```
import javax.servlet.*;
import javax.servlet.http.*;
```



```
import java.io.*;

public class ThreadServlet extends HttpServlet {
   int i;
   public void service(HttpServletRequest req, \
                       HttpServletResponse res) throws IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      synchronized(this) {
         try {
            Thread.sleep(5000); // 5 secondi
         } catch(InterruptedException e) {
            System.err.println("Interruzione");
         }
```

```
        }
        out.print("<h4>Finished " + i++ + "</h4>");
        out.close();
    }
}
```

Se la parte da sincronizzare viene comunque eseguita in ogni caso, si può mettere la parola chiave `synchronized` prima del metodo `service`.

Capitolo 7-CMS e Web Frameworks

venerdì 22 aprile 2022 12:48

- Molte applicazioni per il web hanno tante caratteristiche in comune con le applicazioni desktop->RIA
- Le **RIA** utilizzano tecnologie lato client in maniera intensiva per l'invio asincrono di richieste HTTP->realizzazione di interfacce utente sempre più simili a quelle desktop
- Sviluppo e diffusione RIA->Web 2.0->batterie incluse->CMS e Web Application Frameworks
- **CMS:** gestire i workflows e lo sviluppo di contenuti in ambienti di lavoro collaborativo e vengono progettati per semplificare la pubblicazione di documenti su web->sono RIA
- **Web Frameworks:** alleggerire il lavoro legato allo sviluppo di applicazioni web, fornendo supporto per le funzionalità tipicamente richieste per la loro realizzazione quali la gestione e la visualizzazione dei dati->sviluppano i RIA

7.1. CMS

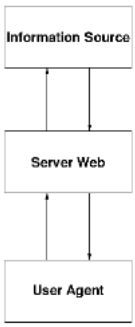
- Sistema gestione dei contenuti
- Organizzare il lavoro sui contenuti di un gran numero di utenti
- Un web CMS funziona su web e prevede la possibilità di accesso principalmente attraverso un browser->soprattutto per la parte di amministrazione può venir previsto anche un diverso client
- Agli utenti non viene richiesta nessuna competenza informatica o web
- La maggior parte dei CMS usa una base di dati relazione per immagazzinare sia i contenuti dei metadati che qualsiasi altro contenuto necessario al sistema->immagazzinati come XML
- Offre supporto alla creazione, modifica e alla pubblicazione dei documenti
- Sviluppa documenti in collaborazione tra più persone
- Importa documenti legacy o esterni nel sistema->i documenti verranno riportati nei formati previsti dal sistema, HTML o PDF->sistema controllo delle versioni
- Offre il retrieval dei documenti->devono essere organizzati in una repository con indicizzazioni opportune in modo da facilitarne l'accesso
- Possono separare i contenuti dalla formattazione permettendo la configurazione degli aspetti grafici
- Controllo degli accessi: definendo diverse categorie di utenti con diversi profili e quindi permessi di accesso
- **Workflow:** supporto al lavoro in collaborazione che segue il percorso di un documento elettronico prevedendo una serie di azioni ad esso associate da parte di diverse persone->gli eventi del workflow devono poter venir segnalati agli interessati in modo asincrono
- **One-to-one marketing:** personalizzazione della pagina per lo specifico utente

7.2. Web framework

- Ne esistono varie tipologie
- Strumento che offre supporto allo sviluppo di applicazioni, mettendo a disposizione una serie di strumenti e soluzioni integrate già pronte all'uso
- Web framework->framework per lo sviluppo di applicazioni specifiche per il web mettendo a disposizione sia strumenti per la programmazione lato server che lato client
- Nascono dalla consapevolezza che le operazioni svolte da uno sviluppatore con la programmazione lato server sono sempre le stesse->fa carico di sistemare e di risolvere questi problemi in modo standard, in modo che lo sviluppatore possa concentrarsi esclusivamente sulle caratteristiche significative dell'applicazione che sta sviluppando

7.2.1. Architettura delle applicazioni web

- Sono multi-tier o n-tier
- I tieri sono un raggruppamento logico di funzionalità->i diversi tier possono stare sulla stessa macchina o su macchine diverse
- Le applicazioni web sono three-tier



- **Controller-logic:** elabora la richiesta del client e si procura i dati da presentargli
- **Presentation logic:** elabora questi dati e li presenta all'utente
- **Business logic:** assicura che i dati siano affidabili prima di usarli per aggiornare la sorgente di informazione o rispondere all'utente
- **Business rules:** regolano l'accesso dei client ai dati e l'elaborazione dei dati da parte dell'applicazione

7.2. Modello MVC e Web Frameworks

- Bisogna adattare il proprio modello di design a quello imposto dal framework utilizzato->solitamente si adotta quella a 3 strati->forte dipendenza e da un alto grado di accoppiamento tra i vari strati e risulta quindi non sufficiente modulare da poter essere sistematizzata in un framework->integrazione

troppo stretta tra livello di presentazione, di elaborazione e l'acquisizione dei dati dalla sorgente di informazione->base di dati
- Obiettivo principale del framework->fornire componenti che possano poi essere combinati in base ad esigenze di sviluppo
- **Pattern MVC**->il modello ha la responsabilità della gestione dell'acquisizione dei dati e del dominio, il controllore dell'elaborazione dei dati e della comunicazione tra le diverse componenti dell'applicazione mentre la vista è la sola responsabile della visualizzazione e presentazione dei dati
- I tre diversi livelli->specifiche funzionalità->specifiche componenti software messe a disposizione dai web frameworks sono:

a. **Model:**centralizza il controllo e l'accesso ai dati, rendendo fruibile al livello di controllo la gestione dei dati->veicolata da un ORM responsabile del mapping tra modello dei dati e modello ad oggetti dell'applicazione
b. **View:** responsabile della composizione e visualizzazione delle risorse spesso rappresentate da pagine HTML, ma anche JSON o XML ed altri ancora
c. **Controller:** ha la responsabilità di acquisire le richieste HTTP, gestire la comunicazione con il modello dei dati e di trasferire i dati alla vista per la compilazione delle risorse->caratteristiche tecniche afferenti a tale livello che comprendono:

i. **URL Routing RESTful:** meccanismo di indirizzamento degli URL supportati dalla applicazione web che rispetti le linee guida della specifica REST
ii. Meccanismi di sicurezza contro attacchi

- I web frameworks offriranno supporto per tutte quelle che sono caratteristiche tipiche di un'applicazione web->session-tracking e autenticazione dell'utente->mettendo a disposizione uno stack di componenti sw->full-stack framworks per riferirsi a quei frameworks che offrono supporto, attraverso componenti propriamente implementate o integrate da strumenti di terze parti per ciascun livello

7.2.3. Micro-stack framework

- Non offrono alcun supporto out-of-the-box per i livelli di modello e vista ma offrono solo l'infrastruttura in grado di gestire e smistare le richieste HTTP per l'applicazione web
- Entrambe le tipologie di web framework aggiungono allo stack software un web server di sviluppo integrato, da impiegare esclusivamente e tassativamente solo per lo sviluppo

7.3.Principi di progettazione e scopi

- Tutti i web framework condividono alcuni principi di design e finalità:

a. **Convention over Configuration:** principio che favorisce l'adozione di un insieme di convenzioni (progettazione e implementazione) piuttosto che la configurazione nel dettaglio dell'applicazione. Lo sviluppatore ha il dovere di configurare l'associazione tra modello e controller->riducendo drasticamente i tempi e l'overhead di implementazione
b. **DRY (Don't repeat yourself):** si tratta di un principio generale che sconsiglia la duplicazione dell'informazione e insiste nel mantenere ogni parte dell'informazione in un unico punto del sistema
c. **Accoppiamento lasco:** i diversi livelli del framework non dovrebbero avere conoscenza approfondita l'uno dell'altro se non dei casi in cui sia necessario
d. **Minor quantità di codice:** lo sviluppo delle applicazioni web deve richiedere meno codice possibile->la maggior parte è a carico del framework
e. **Sviluppo veloce:** velocizzare lo sviluppo su web, in particolare evitando o riducendo il più possibile gli aspetti più noiosi e ripetitivi assicurando meccanismi di gestione già sistematizzati e testati.

Capitolo 8-WebUML

sabato 23 aprile 2022 11:02

8.1. Web engineering

- La progettazione di sistemi basati su web ha una serie di peculiarità:

a. Requisiti instabili
b. Più interazione con i committenti
c. Sviluppo->forti pressioni sui tempi->compressione dello scheduling
d. Tecnologie cambiano con frequenza
e. Non si sfrutta a pieno il paradigma ad oggetti
f. Sviluppati da team eterogenei
g. Non esistono strumenti di sviluppo solidi come in ambito desktop

8.2. Web Engineering: UML estensione Conallen

- Molto ad alto livello, i istemi basati su web non sono altro che un tipo particolare di sistemi client-server->elementi principali: client, server e rete di connessione.
- La maggior parte dei sistemi basati su web considerano utenti anonimi che non hanno abilità o conoscenze informatiche->interfaccia intuitiva e dal comportamento predicibile->use case
- Si utilizza un linguaggio UML

8.2.1. Peculiarità dei sistemi basati su web

- Prevede meccanismi per estenderlo
- Struttura che va usata con attenzione->estensioni introdotte con superficialità possono facilmente condurre a problemi di inconsistenza dei modelli
- Fattore chiave per una buona modellazione->scelta del corretto livello di astrazione e dettaglio
- Gli schemi architetturali di base per le applicazioni web usano le pagine web come elemento di base->poiché uno degli obiettivi nella fase di design è catturare tutte le componenti del sistema, risulta cruciale catturare le pagine web come elementi principali e utilizzarle insieme alle classi e alle altre componenti del sistema
- Le pagine client->artefatti che possono essere considerati come ogni altra interfaccia utente in un sistema
- Come si modellano le pagine server? Questo tipo di pagine interagisce con le risorse lato server prima di essere inviata al client come interfaccia utente completa. Non essendo un approccio ad oggetti->UML non riesce a modellare bene questo importante aspetto dei sistemi web
- Si costruisce un'estensione di UML con:

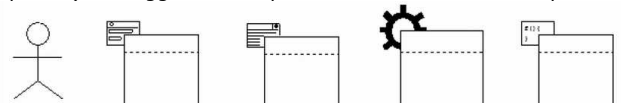
a. **Stereotipi**->classificare i model element di UML. Sia S uno stereotipo del modello M, S conderva tutte le caratteristiche di M ma può inoltre soddisfare dei vincoli addizionali e avere dei nuovi tagged values-><<S>>
b. **Tagged values**->rappresentano un'estensione delle proprietà associate ad un elemento del modello->definizione di una nuova proprietà che può essere associara ad un elemento del modello->(Tag,Value)
c. **Vincoli o constraints impliciti** che modificano la semantica->estensione della semantica del linguaggio->consentono di rappresentare fenomeni che altrimenti non potrebbero essere espressi con UML

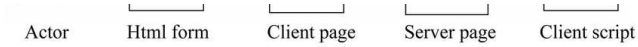
8.2.2. L'estensione di Conallen

- due stereotipi: la pagina server e la pagina client->La prima contiene metodi e variabili relativi allo scripting server-side; la seconda, invece, contiene elementi relativi alla formattazione, allo scripting client-side
- introduciamo anche la site map, un'astrazione dell'insieme delle pagine Web e dei collegamenti di navigazione all'interno del sistema.
- Occorre modellare le pagine, i collegamenti (link) tra di esse, il contenuto dinamico necessario alla loro costruzione e il contenuto dinamico che arriva fino al client.
- Bisogna modellare anche la business logic
- Occorre ora far corrispondere ciascuno di questi quattro elementi dell'applicazione ad un diverso elemento del modello.

a. **Hyperlinks:** associazioni tra elementi.
b. **Pagine:** classi (con tre aree: nome, attributi e operazioni).
c. **Scripts:** operazioni nella classe.
d. Variabili definite con scope di pagina: attributi della classe.

- Ma cosa facciamo se, come succede di solito, una pagina contiene sia scripts lato server (ad esempio, JSP) che lato client (ad esempio, applet o JavaScript)? -> stereotipi che ci permettono di associare un nuovo significato ad un elemento del modello, tagged values, ovvero coppie chiave-valore da associare ad elementi del modello e constraints che permettono di definire delle regole necessarie a che il modello sia ben formato
- **La Implementation View o Component View** descrive i componenti del sistema e le relazioni che intercorrono tra loro->corrispondono alle pagine Web e ai relativi collegamenti, dando origine ad una site map.
- Il comportamento di una pagina Web sul server è completamente diverso che sul client: può accedere alle risorse del server nel primo caso; al browser e alle sue funzionalità nel secondo->Ciascuno di questi due aspetti di ogni pagina verrà quindi modellato separatamente, e la relazione tra le due viene stereotipata come build->Ovviamente mentre ogni pagina lato client può venir costruita da al più una pagina lato server, una pagina lato server può costruire anche più pagine lato client
- **Link**-> ha sempre origine da una pagina lato client e punta ad un'altra pagina che può essere sia lato client che server (non considerare quest'ultima opzione)-> i tagged values permettono di associare dei parametri ai link





- **SERVER PAGE:** Si tratta di una pagina web contenente script eseguiti dal server, che interagiscono con risorse lato server come database, business logic, sistemi esterni. Le operazioni della classe rappresentano le funzioni dello script, gli attributi rappresentano le variabili visibili nello scope della pagina. È soggetta all'unico vincolo di avere relazioni soltanto con oggetti sul server. I tagged values di una pagina server descrivono lo scripting engine
- **CLIENT PAGE**: Si tratta di una pagina direttamente rappresentabile dallo user agent, quale ad esempio il browser. Può contenere script che sono interpretati dallo user agent. Le operazioni corrispondono alle funzioni degli script, mentre gli attributi corrispondono alle variabili. Una client page può avere associazioni con altre pagine, client o server. Non è soggetta a nessun vincolo e ammette i seguenti' tagged values:

a. **Titolo-tag :** il titolo della pagina è mostrato dal browser
b. **Base-tag :** URL di base per dereferenziare relativi URLs
c. **Body-tag :** l'insieme degli attributi per l'elemento <body> che impostano il background e gli attributi di testo di default.

- **PARTIZIONAMENTO LATO CLIENT E LATO SERVER:** Questa attività richiede di individuare le server page e le client page e di stabilire le relazioni tra loro e con gli altri oggetti del sistema. Le due attività principali nel design di applicazioni Web sono significativamente differenti dal design di altri sistemi software: Partizionamento degli oggetti lato client e lato server e definizione delle pagine web come interfacce utenti. Un'architettura Thin web client tutti gli oggetti sono lato server mentre un'architettura Fat web client, in gran parte, oggetti persistenti, oggetti contenitori, oggetti condivisi, e oggetti complessi appartengono al lato server
- **LINK:** tale stereotipo è definito per un'associazione tra pagine client e altre pagine anche se il link collega due pagine client, il server viene sempre coinvolto perché il link presuppone una richiesta HTTP al server.
  Un'associazione di tipo link può essere sia unidirezionale che bidirezionale, nel caso ci siano due link che collegano le due pagine nelle due direzioni. Corrisponde ad un'associazione UML e non viene rappresentato da alcuna icona. Rappresenta un puntatore tra una client page e un'altra pagina. Ha un unico tagged value per i parametri: una lista di nomi di parametri che dovrebbero essere passati insieme al momento della richiesta per la pagina destinazione del link. Un'altra importante associazione è quella che collega una pagina server con la pagina client che essa produce in uscita (build).
- **REDIRECT:** trasferimento del controllo da una pagina ad un'altra. In questo caso sia la sorgente che la destinazione possono essere client pages o server pages. Se l'origine è una client page, la pagina destinazione verrà automaticamente richiesta dal browser, allo scadere di un determinato intervallo di tempo. Non ha nessun vincolo. Ha come tagged value il delay che dà l'ammontare di tempo che la client page dovrebbe aspettare prima del redirect alla prossima pagina.
- **PROGETTO DELLE SINGOLE PAGINE WEB:** Una volta che sono state identificate le Web page, le principali interazioni tra oggetti e web page, e le responsabilità di ogni componente, ha inizio la definizione di ogni singola pagina. Le form rappresentano una componente caratteristiche di moltissime pagine client e quindi bisogna chiedersi come rappresentarle. Contengono attributi addizionali rispetto alle normali pagine. È possibile che una pagina contenga più form. Un form può essere rappresentato come una classe i cui attributi sono i campi del form. La relazione tra pagina e form è un'aggregazione. Lo stereotipo della relazione tra il form e la pagina server che ne utilizza i dati è un submit. Gli attributi di questa classe sono rappresentati dai campi di input dell'elemento. n form non possiede operazioni. Nessun vincolo. Il metodo (ad esempio, GET o POST) usato per sottomettere i dati nella action URL viene rappresentato come tagged value
- **SUBMIT:** Si tratta di un'associazione che associa un form ad una server page, alla quale il form invia i dati. Poi sono elaborate le pagine server che utilizzano le informazioni nel form sottoposto. Non ha nessun vincolo, ma un tagged value per i parametri: una lista di nomi di parametri che dovrebbero essere passati con la richiesta della pagina destinazione dell'associazione.
- **FRAMESET:** Viene rappresentata come una classe contenitore di pagine web multiple. L'area viene divisa in frame, ed ogni frame viene associato ad al più un **<<target>>** (anche nessuno). Il contenuto di ciascun frame può essere una web page o un altro frameset. Essendo una client page, può avere attributi e operazioni, che sono definiti in maniera analoga Nessun vincolo.

a. **Tagged value: Rows :** il valore dell'attributo della riga del tag **<<frameset>>**.
b. **Cols :** il valore dell'attributo della colonna del tag <<frameset>>

- **LOGICHE LATO CLIENT:** I ClientScript Object (ad esempio, file .js) vengono rappresentati da una classe. La semantica di questo stereotipo è una collezione di scripts client-side che esistono in file e sono incluse in richieste da parte di un client browser. Questi oggetti sono spesso accomunati da funzioni usate per le applicazioni. Non hanno né vincoli né tagged values.

Capitolo 9-JSP

lunedì 25 aprile 2022 12:25

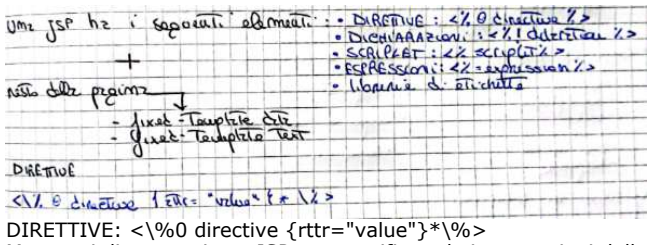
9. JSP: Java Server Page:
- Programmazione in Java lato server
- Il principale strumento per lo sviluppo di applicazioni web in Java insieme alle servlets
- Elezione di metodi dinamici e indipendenti delle piattaforme per la creazione di applicazioni basate sul web
- Offre diversi vantaggi rispetto la CGI (nonostante alcune volte hanno lo stesso scopo dei programmi implementativi che utilizzano la CGI):

a. Prestazioni migliori->permette di incorporare elementi dinamici nella pagina HTML stessa
b. Vengono sempre compilate prima di essere elaborate dai server
c. Possono essere utilizzate in combinazione con le servlets che gestiscono la logica aziendale, il modello supportato dai motori di template di servlet

- Permettono di implementare pagine in cui le istruzioni java e il template statico sono organizzati in modo più chiaro e più facile da mantenere
- La compilazione avviene in modo automatico le prime volte che arriva una richiesta per la pagina e ogni volta che essa viene modificata ma non occorre ricompilare se le modifiche riguardano solo il template statico
- Pagina JSP->pagina di testo formata da:

a. Template statico (HTML, SVG, WML, XML)

- Estensione .jsp per indicare al server il tipo di elaborazione di richiesta->una pagina JSP viene richiesta mentre essa viene tradotta in una servlet->viene poi messa in esecuzione dal servlet engine->si aggiunge il class loader e la si esegue->viene prodotta una pagina web da mandare al browser per essere utilizzata->la servlet può creare oggetti necessari alla computazione e scrivere le corrispondenti stringhe su uno stream di uscita che passa alla risposta HTTP
- Due tipi di errori: durante la traduzione e durante l'elaborazione della richiesta



- DIRETTIVE: <\%0 directive {rttr="value"}*\%>

a. Messaggi di meccanismo JSP per specificare le impostazioni della pagina->le direttive sono indicizzazioni o istruzioni per il contenitore da seguire quando si traduce un JSP in un servlet
b. 3 direttive: page (specificare le impostazioni della pagina), include (includere contenuti da altre risorse) e taglib (specificare librerie di etichette personalizzate)

PAGE:
<\%@ page language="java" \%>
<\%@ page session="true" import="java.util.*" \%>
INCLUDE:
<\%@ include file = "clock.jsp" \%>
TAGLIB:
<\%@ taglib uri = "advjhtp1-taglib.tld" prefix = "advjhtp1" \%>

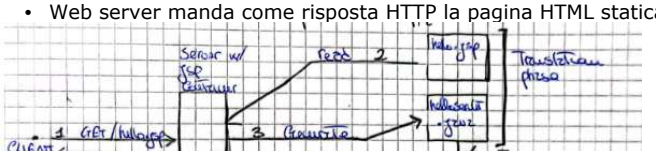
- Oggetti impliciti:

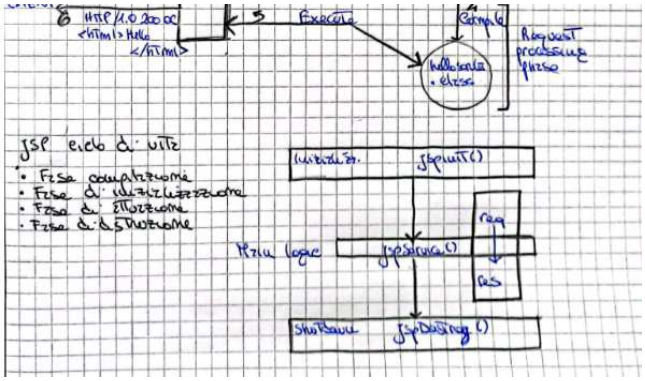
a. Application: il contenitore all'interno del quale viene eseguito la servlet
b. Scope di pagine (config, exception, out, page, pageContext, resp)
c. Scope di richiesta (request)
d. Scope di sessione (session)

- Elementi di scripting: possono essere usati non solo per produrre una pagina dinamica ma anche per produrre pagine statiche da restituire solo se sono soddisfatti determinati requisiti. Sono componenti

a. Scriptlets: <%…%> le istruzioni contenute in uno scriptlet vengono messe in esecuzione durante l'elaborazione della richiesta HTTP
b. Commenti: <%-…-%> JSP, <!-…-!> XHTML oppure commenti in linguaggio di scripting
c. Espressioni: <%=…%>
d. Dichiarazioni: <%!...%>
e. Sequenze di escape:<%,%>, ', ", \

JSP:
- Traduttore->servlet->risposta HTTP->UA
- UA manda richiesta HTTP al server
- Il web server riconosce che si tratta di una richiesta per una pagina JSP e passa la richiesta al motore JSP
- Traduzione in servlet
- Servlet engine JSP fornisce una classe eseguibile e la richiesta originale viene passata al motore servlet
- Viene caricata ed eseguita la classe servlet. Durante l'esecuzione il servlet genera un output sottoforma HTML
- Web server manda come risposta HTTP la pagina HTML statica





Paragrafo 9.1, detto dalla professoressa per la prova intercorso:

- Principale strumento per sviluppare applicazioni web in Java
- Problema servlets->preparazione di una pagina da mandare al client che richiede delle istruzioni intricate in cui il formato utilizzato per la presentazione della pagina risulta immerso in istruzioni di stampa->introdurre errori e bisogna ricompilare la servlet per ogni modifica
- JSP permettono di implementare pagine in cui le istruzioni Java e il template statico convivono->chiaro, leggibile e facile da manutenere
- Compilazione automatica la prima volta che arriva una richiesta per la pagina e ogni volta che la pagina viene modificata->non occorre ricompilare se le modifiche riguardano solo il template statico
- Una JSP è una pagina di testo formata da: un template statico espresso in un qualsiasi formato testuale (HTML, SVG...) e le istruzioni java
- Estensione .jsp indica al server il tipo di elaborazione richiesta->la prima volta che una pagina JSP viene richiamata essa viene tradotta in una servlet->messa in esecuzione dal servlet engine che carica la classe con un class loader e la esegue->viene prodotta una pagina web da mandare al browser per essere visualizzata->la servlet può creare oggetti necessari alla computazione e scrive le corrispondenti stringhe su uno stream di uscita che passa alla risposta HTTP
- Due tipi di errori: durante la traduzione oppure durante la richiesta
- Una pagina JSP ha in più alcuni elementi:

a. **Direttive:** <%@ directive %>
b. **Dichiarazioni**: <%! declaration %>
c. **Scriptlet:** <% scriptlet %>
d. **Espressioni:** <%= expression %>
e. **Librerie di etichette**
f. Resto della pagina-> fixed-template data oppure fixed-template text

Capitolo 10-Session tracking

martedì 26 aprile 2022 14:18

10.0.1. Autenticazione richiesta all'utente

- Affinché il browser rimane aperto, username e password vengono salvati sul client in modo che l'utente non debba digitarli tutte le volte che viene inoltrata una richiesta
- Tramite servlet si può accedere all'informazione della sessione quando usa un meccanismo di autenticazione via HTTP->si usa il metodo getRemoteUser() che fornisce alla servlet l'username dell'utente dopo che ha fatto il login fornendolo insieme alla password

```
String name = request.getRemoteUser();
if(name == null) {
  // message: page is protected
} else {
  String[] items = request.getParameterValues("item");
  if( items != null) {
    for( int i=0; i < items.length; i++ ) {
      addItemToCart( name, items[i] );
    }
  }
}
```

E per elencare gli oggetti nel carrello:

```
String name = request.getRemoteUser();
if(name == null) {
  // message: page is protected
} else {
  String[] items = getItemsFromCart(name);
}
```

- Vantaggi:

a. facile da implementare in quanto il server protegge determinate pagine e può poi identificare ogni client-> usando ad esempio getRemoteUser() per le servlet
b. Il meccanismo delle autenticazioni funziona anche se l'utente accede al sito da macchine diverse
c. Continua a funzionare anche se l'utene esce dal sito o addirittura esce dal browser prima di tornare a concludere la transazione

- Svantaggi:

a. La richiesta di registrazione si adatta solo a determinate situazioni, in tutti gli altri casi occorre un session tracking anonimo
b. Un utente può, in ogni momento, mantenere attiva una sola sessione dato un sito

10.0.2. Campi nascosti nelle form

```
// L'utente puo' aggiungere oggetti al carrello o uscire
// Gli oggetti correnti vengono aggiunti come campi nascosti, in
// modo da conservarli per la prossima richiesta
out.println("<form action=\"/servlet/ShoppingCart\"
            method=\"post\">");
if(items != null) {
  for(int i=0; i < items.length; i++) {
    out.println("<input type=\"hidden\" name=\"item\" value=\""
                + items[i] + "\">");
  }
}
```

- Nei casi in cui l'informazione da passare avanti e indietro ad ogni richiesta è di dimensione non trascurabile->conviene salvare l'informazione direttamente nel server e passare solo un identificativo che permetta di identificare la sessione
- Vantaggi:

a. Sono supportati dai browser più diffusi
b. Non richiedono nessuna particolare funzionalità da parte del server
c. Sono anonimi

- Svantaggi:

a. Funziona solo quando lo scambio prevede una seqauenza di schede da riempire generate dinamicamente
b. Fallisce nel caso di errore che provoca la chiusura del browser

10.0.4. Cookies

- Cookie pezzo di informazione testuale mandata dal server web al browser che può in seguito venire ripassata dal browser al server
- 20 cookies per sito e almeno 300 per utente->si possono anche imporre limiti superiori alla dimensione di ogni cookie di 4k
- Costruttore: public COokie (string name, string value) dove il nome identifica il cookie mentre il valore rappresenta l'informazione associata al cookie
- Per mandare un cookie da una servlet ad un client: public void HttpServletResponse.addCookie(Cookie cookie)->altri cookies possono essere aggiunti con lo stesso metodo ma prima di ogni altro contenuto
- Per estrarre i cookies da una richiesta di un client una servlets usa public Cookie[] HttpServletRequest.getCookies()

```
// per spedire un cookie
Cookie cookie = new Cookie("ID","123");
response.addCookie(cookie);

// per estrarre i cookies
Cookie[] cookies = request.getCookies();
if(cookies != null) {
```

```
  for(int i=0; i<cookies.length; i++) {
    String name=cookies[i].getName();
    String value=cookies[i].getValue();
  }
}
```

- Public void setDomain(String pattern)->restrizioni sul dominio per default->dominio da cui sono state spedite
- Public void setPath(String url)->specifica il dominio e deve comprendere la servlet che ha creato il cookie
- Public void setComment(string comment)->con lo scopo del cookie->il browser può visualizzarlo o no

```
Cookie[] cookies = request.getCookies();
if(cookies != null) {
  for(int i=0; i<cookies.length; i++) {
    if(cookies[i].getName().equals("sessionid")) {
      sessionid = cookies[i].getValue();
      break;
    }
  }
}}

// Se l'identificativo di sessione non e' stato mandato,
// generarne uno; includerlo nella risposta per il client
if(sessionid == null) {
  sessionid = generateSessionId();
  Cookie c = new Cookie("sessionid",sessionid);
  response.addCookie(c);
}
```

- Il problema più importante coi cookies è che non sempre il browser li accetta
- I cookie vengono eliminata alla fine della sessione del browsing->per impostare una durata maggiore si usa il metodo setMaxAge che dà il numero di secondi dopo i quali il cookie verrà eliminato
- **IMPORTANTISSIMO:** i browser inviano i cookies esclusivamente allo stesso dominio memorizzato nel cookie