Protocolli e standard - Introduzione alle reti di calcolatori

Architettura Client - Server

E' un'architettura utilizzata nelle applicazioni distribuite. Il server è il sistema che offre i servizi e i client sono i sistemi che li utilizzano.

Reti

Una rete di calcolatori è un insieme di calcolatori collegati fisicamente in grado di condividere risorse e servizi e di scambiarsi messaggi.

Collegamenti e topologie

1. Collegamenti in rame
2. Fibra ottica
3. Ponti radio (Wi-Fi, 3G, 4G, 5G, bluetooth...)

Livello di distribuzione

- LAN - Local Area Network (1km)
- MAN - Metropolitan Area Network (100km)
- WAN - Wide Area Network (1000km)
- GAN - Global Area Network (10000km)
- PAN - Personal Area Network (10mt)
- BAN - Body Area Network (1mt)

Modalità di connessione

1. Commutazione di circuito
2. Commutazione di pacchetto

Protocolli Un protocollo è un insieme di regole per la comunicazione tra calcolatori

Protocollo TCP/IP: ad ogni macchina è associato un indirizzo IP:

- IPv4: 32 bit, 2^{32} macchine collegabili
- IPv6: 128 bit, 340 miliardi di miliardi di miliardi di macchine collegabili

L'IPv4 è strutturato in 2 parti, l'identificatore della rete (NET ID) e l'identificatore del calcolatore (HOST, ultime due cifre).

La subnet mask è il numero che specifica quale parte dei numeri di un IP contiene il NET ID.

DNS

- **Domain Name System**: servizio che associa un nome ad un IP
- **Domain Name Server**: macchina che offre il servizio

Servizi internet

1. smtp: invio posta elettronica
2. pop/imap: ricezione posta elettronica
3. http: trasferimento risorse web
4. ssh: terminale remoto
5. ftp: trasferimento file

Data la possibilità di un server di offrire più servizi contemporaneamente, questi sono in ascolto su una **porta**.

Pila TCP/IP

La comunicazione avviene attraverso lo scambio di messaggi ad alto livello e ogni strato della rete si rivolge a quello inferiore (nella trasmissione, a quello superiore nella ricezione).

Livelli:

1. Livello di trasporto: TCP
   - Orientato alla connessione, affidabile
   - Il messaggio è diviso in datagrammi (pacchetti)
2. Livello di rete: IP
   - Commutazione di pacchetto, non affidabile
   - Instradamento (routing) dei pacchetti verso la destinazione
3. Livello fisico
   - Vari protocolli (in base alla tecnologia di trasmissione)

Architettura client server

Le 3 idee fondamentali del concetto di internet:

1. **HTTP**: protocollo a livello applicativo
2. **URL**: sistema di indirizzamento
3. **HTML**: linguaggio per i documenti

Terminologia e architettura

Una **risorsa** è una qualsiasi informazione (file, dato) accessibile su un server, il server è colui che fornisce le risorse al client, che le richiede.

Esistono sono fondamentalmente 3 macro tipologie di architetture per applicazioni su web:

1. Siti statici
2. Applicazioni web a 3 livelli
3. Applicazioni client-server con API

Contenuti statici

Il server HTTP *serve* direttamente i file al client (solitamente un browser), le tecnologie utilizzate sono statiche. L'architettura è incentrata sui contenuti.

Applicazione a 3 livelli

Il server HTTP comunica con i client e con lo strato applicativo, questo è scritto in un linguaggio lato server (J2EE, Python, PHP) e si occupa di gestire le richieste, dialogare con il database e creare la risposta. La soluzione ha un problema di scalabilità orizzontale perché lavora sulle sessioni a livello del server e quindi scala bene verticalmente (aumentando le risorse) ma non orizzontalmente (duplicando l'applicazione per smistare il traffico su più macchine) e questo si traduce in alti costi di distribuzione su cloud.

Applicazione con API

Una parte della logica applicativa è spostata sul client, con tecnologie di frontend (nella maggior parte dei casi in Javascript) che mantengono lo stato della sessione ed effettuano richieste al backend stateless. Questo è formato da API implementate secondo lo standard REST e gestisce le transazioni sul DB. Questa modalità consente di alleggerire il server da una parte del lavoro affidata al frontend e consente di scalare facilmente orizzontalmente riducendo i costi di deployment.

Server Web

- Apache HTTP Server (open source, il più diffuso)
- Microsoft Internet Information Services
- Google Web Server (versione di apache modificata)
- nginx (open source, bassa impronta di memoria, in rapida diffusione)

I principali server applicativi per J2EE sono Apache Tomcat, Jetty e Netty

Il server web offre una serie di servizi:

- HTTP verso il client (con autenticazione e autorizzazione)
- Gestione delle risorse sul file system
- Gestione delle applicazioni
- Logging
- Caching

Browser web

- Effettua richieste HTTP

- Renderizza le risposte
- Effettua caching locale

Deployment di un sito statico

Deployment su server dedicato

In questo caso il stio va installato su un server in rete dell'organizzazione, è necessario rendere il server accessibile dall'esterno, per fare ciò bisogna assicurarsi di avere un indirizzo IP pubblico statico. E' necessario inoltre istruire il router a inoltrare la richiesta (**NAT**) in ingresso dall'esterno verso l'indirizzo e la porta del server su cui gira il server http. Poi bisogna acquistare un dominio e puntare i DNS del dominio sull'IP pubblico della rete.

La soluzione permette una gestione completa ma è svantaggiosa perché obbliga l'organizzazione a dover gestire tutte le problematiche di amministrazione del server come l'installazione, l'aggiornamento, la gestione della sicurezza, la gestione delle copie di backup, il disaster recovery, il downtime, la velocità di collegamento...

Hosting presso provider cloud

Lo spazio è acquistato presso un provider (AWS, Google Cloud, Microsoft Azure) che nella maggior parte dei casi offre il servizio di acquisto del dominio e collega automaticamente i DNS. A questo punto non resta che caricare i file del sito web sulla macchina remota utilizzando protocolli ftp o sftp. # Risorse e URI

MIME Types

Acronimo di **Multipurpose Internet Mail Extensions**

Sono stringhe per la descrizione del formato delle risorse, utili nelle pagine html o nelle richieste così come nella posta elettronica.

Ogni volta che client e server scambiano risorse indicano nel messaggio il MIME type.

Il browser gestisce le risorse ricevute in base al MIME type di queste

Es.

- text application/msword
- image image/gif
- messagge application/json

URI

Uniform Resource Identifier

Modo per rappresentare indirizzi: *<protocollo>:<indirizzo>*

Protocolli:

- http/s
- mailto
- ftp
- file

Forma generica di un url HTTP

`http://<server>[:<porta>][/<percorso>][?<query>]`

La querystring è una lista di coppia `nome=valore` separate dalla `&`.

Richieste

- Richieste dell'utente (inoltrate attraverso il browser o client)
- Richieste di collegamenti (click su link da pagina html)
- Richieste implicite (richieste del browser per renderizzare elementi come immagini o css)

REFERER: URI della pagina da cui si origina una richiesta HTTP. Parametro bloccato da alcuni browser per motivi di privacy perché consentirebbe ai server di ricostruire parzialmente o completamente la cronologia dell'utente.

File System

Il file system di un server http è virtuale. L'organizzazione non corrisponde all'organizzazione dei file sul disco, la root `/` può essere montata su una qualsiasi cartella del disco, es. `/home/sito/` o `F:\sito\`, dalla radice è visibile tutto il contenuto della cartella.

E' possibile montare altre porzioni del filesystem attraverso alias, ad esempio si potrebbe montare la cartella `C:\Users\utente\` in `/utente`.

Il meccanismo dell'alias viene utilizzato anche nelle applicazioni e questo è il motivo principale per il quale *un URL non corrisponde necessariamente ad una risorsa fisica* ma può essere il punto di accesso per una risorsa generata dinamicamente da un componente software. # Protocollo HTTP

Il protocollo nasce nel 1991, attualmente la vesione 1.1 è supportata da tutti i browser, dal 2015 esiste la HTTP/2.

Il concetto principale del protocollo HTTP è la **transazione**, una transazione è uno scambio di messagi tra client e server:

- apertura connessione
- il client invia una richiesta
- il server invia una risposta
- chiusura connessione

Il protocollo non è orientato alle connessioni, questo significa che per ogni nuova transazione ci sarà una connessione differente e le transazioni sono tra di loro indipendenti.

Struttura dei messaggi

La struttura generale e comune sia a richiesta che risposta è:

```
<linea iniziale>
<intestazioni opzionali>...
<intestazione>:<valore>
<intestazione>:<valore>
<linea vuota>
<corpo, opzionale>
```

Intestazioni

Contengono metainformazioni sulla richiesta o risposta, ad esempio lo User-Agent (il sistema utilizzato), il Referer, il Content-Type e altre informazioni opzionali

Richiesta

Nella richiesta la linea iniziale contiene il percorso della risorsa a cui si vuole accedere. Il corpo è vuoto o contiene parametri della richiesta.

La linea iniziale ha il formato: `<metodo> <percorso> HTTP/1.0`

I metodi di HTTP 1.0 sono GET e POST

GET è il metodo standard, il corpo della richiesta è vuoto e i parametri sono passati nella querystring

Es. `GET /bollo.php?targa=A123 HTTP/1.0`

POST è utilizzato per comunicare con i servizi, i parametri sono passati nel corpo della richiesta e quindi sono nascosti, questo consente di bypassare il limite di lunghezza delle querystring.

es.

```
POST /bollo.php HTTP/1.0

targa=A123
```

Risposta

Nella risposta la linea iniziale contiene l'esito della richiesta. Il corpo contiene il contenuto della risposta se previsto.

La riga iniziale ha il formato: `HTTP/1.0 <codice numerico> <descrizione>`

Classi di codici:

- 1xx: messaggio informativo
- 2xx: richiesta esaudita con successo
- 3xx: redirect
- 4xx: errore lato client
- 5xx: errore lato server

Esempi di codici

- 200 ok
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal server error
- 503 Service Unavailable

Autenticazione e autorizzazione

L'autenticazione è la procedura secondo la quale l'utente si identifica al server fornendo delle credenziali che il server confronterà nel proprio archivio (solitamente un db).

L'autorizzazione è la possibilità di un utente appartenente ad una determinata classe di utenti di eseguire un'operazione o accedere ad una risorsa.

HTTP 1.1

Novità e miglioramenti introdotti

Connessioni persistenti Con l'http 1.1 sulla stessa connessione TCP possono avvenire più transazioni, la connessione viene chiusa o dal server dopo un certo timeout, o dal client con l'intestazione `Connection: close`.

Host Virtuali IP e porta non bastano ad identificare un server, questo consente ai provider di utilizzare lo stesso ip e indirizzare la connessione su più server, il client deve specificare l'intestazione **Host** con il nome del server.

Autenticazione Digest Nell'autenticazione le password non vengono trasmesse, il server invia al browser una stringa **nonce** (number used once) e il browser risponde con il nome utente e un valore crittografato basato su utente, password, percorso e nonce, il server confronta questo valore con quello calcolato e stabilisce se l'autenticazione è valida.

Il meccanismo non è sicuro perché la richiesta è intercettabile.

SSL Secure Socket Layer è un protocollo di trasporto che utilizza un meccanismo a chiave pubblica per crittografare tutti i messaggi trasmessi. un server HTTP gestisce le richieste HTTPS sulla porta 443.

Metodi di accesso aggiuntivi HTTP 1.0 introduce una serie di metodi di accesso aggiuntivi per facilitare le operazioni sulle risorse:

1. PUT
2. DELETE
3. OPTIONS
4. TRACE
5. UPGRADE

Per la gestione del caching si differiscono i metodi in **idempotendi** e **non idempotenti**. Un metodo si dice idempotente se esecuzioni successive di una richiesta producono risultati identici alla prima.

Sono metodi idempotenti GET, PUT e DELETE.

I metodi per i quali è consentito il caching sono GET e HEAD, nessuno degli attori coinvolti nelle richieste effettua caching dei metodi POST, PUT o DELETE.

Un problema da gestire è l'invio ripetuto di richieste (per esempio perché l'utente clicca due volte sul pulsante o ricarica una pagina erroneamente o perché la vista non risponde) che può causare duplicazioni di dati o errori lato server.

HTTP/2

Supportato dal 97% dei client ma solo il 50% dei server.

La principale novità è che in un'unica connessione è possibile caricare più risorse in parallelo e il server può restituire dati anche non richiesti. Ad esempio se il client richiede una pagina html il server può restituire autonomamente la pagina e le risorse (jpg, css, js) ad essa collegate.

L'utilizzo a priori di HTTP/2 non è consigliato per la mancata copertura del 100% dei client ma può essere utile nella comunicazione tra microservizi di backend.

Stato e Sessioni

Cookies

Uno dei meccanismi per ovviare al problema di mancanza di stato di HTTP sono i cookies, si tratta di coppie chiave valore che il server aggiunge alla risposta. Il client (se accetta il cookie) si impegna a restituirlo con le successive richieste per consentire al server di collegarle alle precedenti.

Intestazioni

- **Set-Cookie** intestazione inviata dal server
- **Cookie** intestazione del client nella richiesta

Un cookie è associato ad un URI ed è valido per tutti gli URI che contengono come prefisso quello a cui è associato.

Un cookie può avere validità di sessione (finchè non viene chiusa la tab) o una scadenza nel tempo (anche infinita).

Tokens

Per gestire le autenticazioni si utilizzano i token

Bearer Token Viene inviato dal server a seguito dell'autenticazione di un client, il server aggiunge all'intestazione (o come attributo) il token (una stringa), che autorizza chiunque la invii nella richiesta ad accedere come l'utente autenticato.

Questo risolve il problema dell'autenticazione ma non dell'autorizzazione, poiché il server non riesce a risalire all'utente e ai suoi permessi.

JWT - Json Web Token I JWT risolvono questo problema, i token sono stringhe crittografate di un oggetto json contenente informazioni sull'utente.

XSS - Cross Site Scripting

Tecnica di attacco che sfrutta gli input non sicuri che vengono poi mostrati in una pagina. Immettendo in un input del codice html o javascript il server lo inserisce nella pagina e questo viene eseguito.

In questo modo è possibile creare dei link (sfruttando parametri passati come get) che eseguono nella pagina codice javascript proveniente da altri domini. Uno script eseguito in questo modo può accedere a cookie, intestazioni e informazioni sulle richieste e può risalire a informazioni sensibili per l'utente a partire da questi.

L'attacco può essere anche persistente se la porzione di codice malevolo viene memorizzata nel DB (ad esempio in un social network o forum) e viene automaticamente eseguita sui client di tutti gli utenti che visitano la pagina anche senza necessità che si clicchi su link.

Protezioni

Il primo e fondamentale rimedio per questo tipo di attacco è la sanificazione degli input che consiste nel rimuovere tag e keywords pericolose riconosciute come codice eseguibile.

Una seconda forma di protezione è offerta dai browser:

Same Origin Policy Definiamo come Origin di una richiesta la parte del referer che corrisponde a protocollo dominio e porta, es. [https, www.unibas.it, 80]

La Same Origin Policy è un insieme di regole che bloccano le richieste Cross Origin originate da Javascript. **Il browser effettua la richiesta ma ne blocca l'accesso alla rispota**.

Questo viene effettuato a livello di richiesta con l'intestazione `Origin: <origine>` del browser.

Per gli scopi di sviluppo e in generale per le nuove architetture basate su microservizi e separazione tra backend (con api) e frontend questo meccanismo costituisce un problema.

Lo standard per controllare le richieste cross origin è il **CORS - Cross Origin Resource Sharing**.

Le richieste XO possono essere effettuate aggiungendo al server 2 richieste http:

- Access-Control-Allow-Origin: <origin>
- Access-Control-Allow-Credentials: true

Ogni richiesta eseguita viene confrontata con quelle indicate nella prima intestazione e se è presente viene accettata correttamente e la prima intestazione viene restituita anche nella risposta.

La seconda intestazione serve per esporre i cookie in una richiesta XO (cosa di default disabilitata dal browser).

Richieste di Preflight Il browser non può sapere se una richiesta XO verrà bloccata finché non la effettua, per velocizzae le cose esistono richieste di preflight, attraverso il metodo OPTIONS consentono di effettuare richieste molto leggere per verificare se il server risponde positivamente ad una richiesta XO.

NON C'E' PREFLIGHT PER LE RICHIESTE GET # Java e HTTP

Il package `java.net` offre un'interfaccia per la comunicazione a basso livello e una per la comunicazione ad alto livello.

Socket

Le socket sono canali di comunicazione tra server e client.

Il client apre una connessione con un oggetto `java.net.Socket` e invia richieste con un OutputStream.

Client

```
Socket socket = new Socket("localhost", 8080);
PrintWriter richiesta = new PrintWriter(socket.getOutputStream());
richiesta.println("Hello");
```

```
richiesta.flush(); // Ripulisci il buffer
BufferedReader risposta = new BufferedReader(new InputStreamReader(socket.getInputStream()))
String linea;
while ((linea = risposta.readLine()) != null) {
    System.out.println(linea);
}
socket.close();
```

Server

```
ServerSocket serverSocket = new ServerSocket(8080);
Socket socket = serverSocket.accept(); // Bloccante fino alla richiesta
BufferedReader flusso = new BufferedReader(new InputStreamReader(socket.getInputStream()));
String richiesta = flusso.readLine();

PrintWriter risposta = new PrintWriter(socket.getOutputStream());
risposta.println("Echo:" +richiesta);
risposta.flush();
socket.close();
```

Le socket sono utlizzate quando si ha necessità di creare connessioni molto veloci e poco strutturate, ad esempio in un videogioco multiplayer o una chat istantanea. In generale per i casi standard sono poco utilizzate.

HttpURLConnection

Classe astratta URLConnection e classe HttpURLConnection

Esempio di utilizzo

```
HttpURLConnection urlConnection = null;
        try {
            URL url = new URL("http://www.google.it");
            urlConnection = (HttpURLConnection) url.openConnection();
            urlConnection.setRequestMethod("GET");
            urlConnection.setConnectTimeout(10000);
            urlConnection.setDoOutput(true);
            PrintWriter writer = new PrintWriter(urlConnection.getOutputStream());
            writer.write("q=Ciao");
            writer.flush();
            urlConnection.connect();
            InputStream risposta = urlConnection.getInputStream();
        } catch (IOException e) {
        } finally {
            urlConnection.disconnect(); // chiudo la connessione nel finally
        }
```

Una libreria alternativa è Apache HttpClient # Tecnologie Lato Client - HTML

La prima versione di HTML esce nel giugno '93 per opera di Tim Berners Lee e Dave Ragget, basato su DTD di SGML, il padre di XML.

L'HTML risulta da subito meno restrittivo di xml, i numerosi problemi che questa libertà causava (dovuti alle varie implementazioni dei browser) richiesero una standardizzazione che inizialmente prevedeva vincoli molto rigidi (XHTML gennaio 2000).

Attualmente lo standard riconosciuto e supportato da tutti i browser è l'HTML5 che non deve essere necessariamente un documento XML ben formattato ma deve essere coercibile in XML. Infatti è il browser stesso che si occupa di tradurre un HTML5 nel XML equivalente e correggere eventuali errori.

In HTML5 è convenzionale introdurre una netta separazione tra struttura e stile (che deve essere gestito attraverso fogli CSS).

Elementi

La struttura di base di un documento HTML5 è:

```
<!DOCTYPE HTML>
<html lang="en">
    <head>
        <title>Titolo</title>
        <!-- Metadati della pagina-->
    </head>
    <body>
        <!-- Corpo della pagina-->
    </body>
</html>
```

2 tipi di elementi

- Block level elements: elementi di blocco, i relativi riquadri cominciano sempre una nuova linea. Es. titoli, div, tabelle, liste, paragrafi
- inline elements: elementi i cui relativi riquadri possono essere disposti lungo la stessa linea. es. testo, span, immagini, link

Il browser contiene un **foglio di stile standard** che utilizza per tutti gli elementi che non hanno altre regole CSS specificate.

Attributi

- **id**: identificatore univoco per un elemento
- **class**: utile per dare regole di stili a più elementi
- **lang**: lingua del contenuto (di solito applicata al tag html)

Le metainformazioni contenute nell'HEAD sono utilizzate per specificare intestazioni HTTP relative alla pagina o per aiutare i motori di ricerca nell'indicizzazione.

```
<meta name="author" content="Autore" />
<meta name="keywords" content="prova, html,..." />
```

> Gli appunti non sono completi, ho tralasciato le parti di sintassi e i tag

CSS

Cascading Style Sheets, nasce con l'boettivo di specificare caratteristiche di formattazione per i riquadri di una pagina XHTML.

Il CSS è basato su **regole**, una regola deve contenere un riferimento al riquadro da formattare (**selettore**), una lista di **dichiarazioni** composte da una **proprietà** e il **valore** da attribuirgli.

```
h2 {
    color: green;
    font-size: 14px;
}
```

Sintassi: `<selettore> { <proprietà>: <valore>; altraProprieta: valore;}`

I commenti in CSS si delimitano con `/* */`

Per fare riferimento ad un URI esiste una sintassi particolare: `url("sfondo.jpg")`

Selettori

Un selettore è un indicatore dell'elemento (o insieme di elementi) html a cui applicare le regole di stile contenute tra parentesi quadre. Esistono diversi modi per selezionare un elemento:

Selettori semplici

- Nome dell'elemento (tag): es. `img { border: 1px; }`
- In base all'attributo id: es. volendo selezionare un div `<div id="esempio">` la sintassi css è `#esempio {}` o `div#esempio {}`
- In base all'attributo classe: `.nomeclasse { }` o `p.nomeclasse {}`, il primo seleziona tutti gli elementi che hanno la classe `nomeclasse`, il secondo solamente i paragrafi.

Pseudo-classi

Le pseudoclassi sono selettori utili per classificare elementi dinamicamente in base a condizioni che si verificano a livello di azioni che l'utente compie sull'interfaccia.

Sintassi: `<selettore>:<pseudo classe>`. Ad esempio se si vuole indicare un comportamento per i link quando vengono attraversati dal mouse si può utilizzare

la classe *hover*:

```
a:hover {
    color: red;
}
```

Selettori contestuali

Sono selettori che consentono di distinguere gli elementi in base alla loro posizione nel DOM, costituiti da un selettore ordinario e un **contesto**: `<contesto> <selettore semplice>`.

La sintassi funziona se tra questi due elementi è presente lo spazio, in caso ci sia la virgola invece questa vale come OR logico, ovvero seleziona entrambi gli elementi separati dalla virgola.

```
div p {} /* seleziona i paragrafi all'interno di un div */
div#main img {} /* seleziona le immagini all'interno di un paragrafo con id main */
p span.prova {} /* seleziona gli span con classe prova all'interno di un paragrafo */
h2.titolo, p {} /* seleziona gli h2 di classe titolo E tutti i paragrafi */
```

Lo spazio seleziona gli elementi "all'interno di", il `>` seleziona gli elementi direttamente figli del contesto: es. `div.home > p` seleziona solo il paragrafo direttamente figlio del div.home.

Lunghezze

- Unità assolute: in, cm, mm, pt
- Unità relative: em, rem, ex, vh, vw, px
- Percentuali
- Parole chiave: es. small, x-small

px è la dimensione in pixels, dipende dalla risoluzione dello schermo. emè la dimensione del font del riquadro ad eccezione del font-size relativo alla dimensione del font per il riquadro del padre.

rem è la dimensione del font dell'elemento body.

vh è l'altezza del viewport, vw è la larghezza del viewport.

Ogni browser ha una dimensione predefinita assegnata a ciascuna delle parole chiave:

- xx-small
- x-small
- small
- medium
- large
- x-large
- xx-large

Colori

I colori si rappresentano in codifica RGB o con le keywords standard di HTML. Un colore RGB è una stringa che contiene le codifiche esadecimali dei livelli di rosso verde e blu: `#<rr><gg><bb>` es `#FF000` è il rosso.

Box Model

Ad ogni elemento HTML corrisponde un riquadro, questo è articolato in vari spazi:

- content
- padding
- border
- margin

Le proprietà relative ai margini sono: `margin-top`, `margin-bottom`, `margin-left` e `margin-right`.

Così come le proprietà relative al padding: `padding-top`, `padding-bottom`, `padding-left`, `padding-right`.

Dei bordi possiamo impostare la larghezza (border-width), il colore (border-color) e lo stile (border-style).

Per l'attributo `margin` è possibile impostare il valore `auto` che centra il box orizzontalmente rispetto al padre.

Di default la dimensione di un box è data da width/height + padding + border.

Per disabilitare queto comportamento è possibile aggiungere la regole `box-sizing: border-box` che rende la dimensione dipendente solo da width e height, il padding e i bordi vengono considerati interni.

Solitamente questa viene impostata come opzione di default perché rende più semplice ragionare sulle dimensioni dei riquadri.

Semantica delle regole CSS

Ereditarietà

Alcune proprietà CSS sono ereditate dai predecessori (ad esempio il font), altre no (es. sfondo). Per sapere se una proprietà è ereditata *inherit* consultare lo standard

Cascata

Il meccanismo nasce per l'integrazione sulla stessa pagina di fogli di stile diversi. Le dichiarazioni hanno regole di precedenza e si applicano in cascata.

Il qualificatore per dare priorità massima ad una dichiarazione è `!important`.

```
p { font-size: 1.5em !important }
```

Algoritmo per la priorità:

- Trova tutti i valori per una proprietà
- Ordina rispetto all'origine, in ordine:
  - utente !important
  - autore !important
  - autore
  - utente
  - standard
- Ordina rispetto alla specificità partendo dai più specifici
  - id
  - classe
  - nome
  - ...
- ordina per ordine di comparizione nel file css (le ultime prima)

Layout

Il flusso di visualizzazione è la disposizione dei riquadri nel viewport, la proprietà display lo influenza.

Valori di display:

- **block**: interruzione di linea
- **inline**: nessuna interruzione di linea
- **none**: non viene visualizzato

Nell'algoritmo tradizionale i blocchi annidati corrispondono a riquadri annidati, gli elementi vengono disposti in base al valore di display.

Con il CSS3 vengono introdotti il layout GRID e FLEX:

Grid

`display: grid`, gli elementi possono essere disposti per righe e colonne analogamente a una tabella.

Flex

Maggiore flessibilità, utile per controllare la disposizione dei riquadri alterando il flusso standard.

Una serie di opzioni personalizzabili:

```
flex-direction: row; /* row, column, row-reverse, column-reverse  DIREZIONE PER DISPORRE GL
flex-wrap: wrap; /* nowrap, wrap, wrap-reverse COMPORTAMENTO QUANDO SI RAGGIUNGE IL LIMITE
justify-content: center; /* per i riquadri:  flex-start, center, flex-end, space-around, sp
```

```
align-content: stretch; /* per le righe:  stretch, center, space-around, space-between PER
```

Position

Posizioni in CSS3:

1. static
2. relative
3. fixed
4. absolute
5. sticky

Static Posizionamento classico secondo l'algoritmo di flusso

Relative Posizionamento rispetto al riquadro padre, con due attributi `top` e `left`, se lasciati a 0 la visualizzazione è identica a static.

Fixed Coordinate specificate come distanza rispetto al viewport, rimane fisso anche se il contenuto della pagina scorre.

Sticky Resta visibile finché il container è visibile.

Normalmente è posizionato in modo relative, quando raggiunge il limite superiore (o inferiore) della pagina, l'elemento resta visibile finché il container non esce dal viewport.

Absolute Coordinate espresse come distanze dal primo riquadro padre che ha una position diversa da static. Se non c'è nessun riquadro genitore con queste caratteristiche il riquadro diventa automaticamente il body e quindi il comportamento è identico al fixed.

Float e clear La proprietà `float` può assumere valori `left` o `right`, fissa un riquadro fuori dal flusso ordinario, l'elemento diventa block e gli altri elementi gli scorrono intorno.

La proprietà `clear` dichiara il comportamento di un elemento vicino a un elemento flottante, può assumere valori `left`, `right` o `both`.

Stacking Context

Oltre alle 2 dimensioni x,y gli elementi possono anche essere posizionati rispetto all'asse z.

Uno stacking context è un gruppo di elementi ordinabili rispetto all'asse z. Di default una pagina html ha un unico S.C. corrispondente al tag html. La

disposizione è in base all'ordine di apparizione dei riquadri, quelli apparsi dopo appaiono sopra gli altri.

L'attributo per modificare il comportamento di default è `z-index`, che accetta come valori `auto` o un indice numerico (positivo o negativo). Più alto è l'indice alto sarà il posizionamento dell'elemento sull'asse z.

L'attributo ha effetto solo su elementi che originano a loro volta uno stacking context (ovvero elementi con position diverso da static). Questo fornisce un esempio di utilità del `position: relative` senza specificare altri attributi.

Media Queries

Regole da applicare solo sotto determinate condizioni del dispositivo che visualizza la pagina. La sintassi è:

```
@media <regola> {
  <regole css>
}
```

Alcune regole sono `print` per la visualizzazione di stampa, `min-width`, `max-width`, `min-height`, `max-height` per la larghezza e altezza del viewport, `orientation` per l'orientamento `min-resolution`, `max-resolution`...

Variabili e import

CSS fornisce alcuni strumenti per facilitare lo sviluppo e l'organizzazione delle regole in progetti di grandi dimensioni:

La direttiva `@import` consente di importare regole di stile da un altro file css. Deve essere la prima dichiarazione del foglio.

Similmente a come si fa in un linguaggio di programmazione è possibile definire delle variabili ed utilizzarle nel foglio di stile, evitando ridondanze:

Sintassi:

```
:root {
  --variabile: 20px; /* Dichiarazione, le variabili hanno la visibilità nell'elemento che l
}

/* Utilizzo */
div {
  padding: var(--variabile); /* Si utilizza l'espressione var per recuperarne il valore */
}
```

Per effettuare calcoli (somme, differenze...) sulle variabili è possibile usare la funzione `calc()`. Es. `calc(var(--variabile) + 1px);`

Responsiveness

Negli anni sono stati sviluppati framework per consentire di sviluppare rapidamente interfacce web responsive. Si tratta di file css e javascript già pronti da inserire nei progetti.

Boostrap

E' un framework open source sviluppato da Twitter. Composto da file css e javascript (utile solo per i componenti complessi), fornisce una serie di classi css da attribuire agli elementi che si desidera formattare e una serie di elementi già pronti da recuperare nella documentazione. Il meccanismo di layout è basato su Flexbox.

La struttura di una pagina in bootstrap è formata da div con classe `.container` o `.container-fluid`, div righe (classe `.row`) e div colonne `.col`.

Ogni riga può contenere 12 colonne, è possibile specificare per ogni div colonna quante righe deve occupare.

Bootstrap include delle media query predefinite per i diversi tipi di schermo:

- Extra Small (xs la dimensione di default per le regoole)
- Small (sm)
- Medium (md)
- Large (lg)
- Extra large (xl)

Attraverso le classi bootstrap e le media query è possibile specificare le dimensioni di una colonna a seconda del dispositivo su cui viene visualizzata con la sintassi `<div class=".col-<dimensione>-<numeroColonne> .col-<altraDimensione>-<altroNumeroColonne> ...">`.

Ad esempio

```
<div class="container">
  <div class="row">
    <div class="col-lg-4 col-md-6 col-sm-12">
    </div>
    <div class="col-lg-4 col-md-6 col-sm-12">
    </div>
    <div class="col-lg-4 col-md-6 col-sm-12">
    </div>
  </div>
</div>
```

Una classe css vale anche per le classi successive (partendo dal basso) a meno che non venga ridefinita.

Le colonne hanno una serie di classi aggiuntive: `grow` `shrink`, `offset`, `order`, `margins`...

Tailwind CSS

TailWind offre un approccio più flessibile di bootstrap: è un insieme di microclassi css che possono essere assemblate per produrre elementi complessi.

A differenza di boostrap è necessario un preprocessore che scansiona i file e produce un file css con le sole classi utilizzate.

Javascript nel browser

Il **Document Object Model** è la piattaforma e l'interfaccia che consente a programmi e scripts di accedere dinamicamente e aggiornare i contenuti, la struttura e lo stile di un documento. Il DOM del browser viene chiamato **BOM**.

In generale il DOM è sia la struttura dei dati (equivalente dell'infoset in XML) sia una API per manipolare l'albero.

Le API del DOM sono fondamentali per costruire applicazioni *rich client* in javascript. Infatti il modello e controllo sviluppati in javascript girano sul client e modificano la vista (attraverso le API del DOM), scritta in html e css.

L'oggetto `window` del BOM permette l'accesso ad informazioni sulla finestra del browser (es. `innerHeight` e `innerWidth`). L'oggetto `window.screen` consente invece di accedere ai parametri dello schermo: `height`, `width`, `availHeight`, `availWidth`.

Javascript può accedere ad una serie di informazioni e interazioni con la finestra del browser es. `window.navigator`, `window.location` (uri), `window.location.assign("")`, `window.alert`, `window.confirm`, `window.prompt`, gli ultimi 3 servono per aprire finestre di interazione con l'utente.

Navigare il DOM

L'oggetto `document` rappresenta la radice del DOM. Attraverso di esso è possibile recuperare, modificare, eliminare e aggiungere nodi. Contiene una serie di metadati accessibili come proprietà:

- `document.lastModified`
- `document.title`
- `document.URL`
- `document.baseURI`
- `document.domain`
- `document.referrer`
- `document.cookie`
- `document.readyState`

I tipi di nodi sono:

- ELEMENT_NODE

- ATTRIBUTE_NODE (deprecato)
- TEXT_NODE
- COMMENT_NODE
- DOCUMENT_NODE
- DOCUMENT_TYPE_NODE

Le proprietà `innerText` e `innerHTML` consentono di modificare rispettivamente il contenuto testuale e il contenuto HTML di un nodo. Per modificare una proprietà di stile di un nodo `n` è possibile usare la sintassi `n.style.proprieta = valore`.

Per la ricerca dei nodi esistono diverse funzioni, le più utilizzate:

- `document.getElementById(id)` seleziona gli elementi con un determinato id
- `document.getElementsByTagName(name)` restitusice una HTMLCollection (lista)
- `document.getElementsByClassName(classe)`
- `document.querySelectorAll(selector)` ricerca sulla base del selettore css

Esistono alcune collezioni predefinite nel dom:

- `document.forms`
- `document.images`
- `document.anchors`
- `document.scripts`

Eventi

Ogni elemento in HTML può essere sorgente di evento e il DOM prevede la registrazione di un gestore di eventi per ognuno.

Esempi di eventi sono: onclick, onload, onmouseover, onchange, onsubmit, onkeyup/down/press.

Esistono 3 modi per registrare un gestore di un evento:

1. Utilizzare la proprietà evento nel codice js: `nodo.onclick = funzione()`
2. Dichiarare il gestore direttamente come attributo in html: `<button type="button" onclick="funzione()">`
3. Tramite una istruzione esplicita di sottoscrizione: `nodo.addEventListener(event, funzione)`.

E' possibile modificare il DOM seguendo 2 approcci:

1. Modificare i nodi esistenti, gestire la dinamicità agendo sull'attributo display dei nodi per visualizzarli/nasconderli
2. Creare ed eliminare nodi quando necessario (più efficace quando è necessario gestire collezioni dinamiche). Metodi `appendChild()` e `removeChild()`

Local Storage e Session storage

Il browser dispone di strutture dati per memorizzare informazioni. La differenza tra local storage e session storage è che il secondo è volatile (viene eliminato alla chiusura del browser).

Si tratta di mappe chiave valore, per memorizzare un dato si utilizza la sintassi `localStorage.setItem(chiave, valore)`, per recuperarlo `localStorage.getItem(chiave)`. Lo spazio è limitato a 5MB per il localstorage, per il sessionstorage l'unico limite è la quantità di RAM assegnata dal OS.

Chiamare API

La funzione standard per effettuare richieste HTTP con Javascript dal browser è la funzione `fetch()`. E' una funzione asincrona che restituisce una promise.

Esempio di chiamata fetch con logging della risposta:

```
fetch("http://example.com").then(
    (response) => response.json()).then(
    (data) => console.log(data));
```

Aspetti Metodologici

La qualità di un sito web si basa sulla qualità dell'interfaccia e la qualità di contenuti e servizi offerti.

Qualità dell'interfaccia:

1. Accessibilità
2. Usabilità
3. Caratterizzazione grafica

L'accessibilità è standardizzata nelle linee guida **WCAG**: Web Content Accessibility Guidelines. Le linee guida hanno 3 livelli di priorità (A, AA e AAA).

Tutte le linee guida hanno come obiettivo il rendere il sito fruibile da qualsiasi utente (anche con disabilità e browser particolari), su qualsiasi dispositivo in modo semplice.

L'Usabilità è il requisito che rende un sito utilizzabile con facilità e soddisfazione da parte dell'utente finale. Dipende dalla correttezza (evitare link non funzionanti, immagini troppo grandi...), dall'organizzazione del sito (in sezioni, pagine, in modo che tutti i componenti siano accessibili) e dalla "leggerezza" del sito, che permette a utenti con connettività limitata di accedervi senza lunghi tempi di attesa.

Servizi Web

Un'applicazione distribuita è composta da moduli che girano su macchine diverse e comunicano attraverso messaggi scambiati via rete.

2 moduli fondamentali:

- **Applicativo di frontend**: riguarda l'interazione con l'utente finale, schermi, interfaccia
- **Applicativo di backend**: riguarda i servizi applicativi offerti (al frontend) e l'interazione con il DBMS. Può essere suddiviso in **microservizi**.

Operazioni CRUD

Sono operazioni tipiche relative alla persistenza di una classe nel database. 4 operazioni

- **Create**
- **Retrieve**
- **Update**
- **Delete**

Remote Procedure Call

Il problema principale è come far comunicare i moduli atraverso la rete, la comunicazione è chiamata Remote Procedure Call (RPC).

Anche per le RPC è necessario stabilire un protocollo di comunicazione e un formato per la serializzazione degli oggetti.

I moduli backend espongono **API**, in particolare Web API.

Un **DTO - Data Transfer Object** è un oggetto creato ad hoc per la serializzazione, differente dall'oggetto nel modello.

SOAP - Simple Object Access Protocol

Lo standard (ormai superato) dei protocolli RPC.

I messaggi sono scambiati con HTTP in formato XML. Il messaggio è chiamato *busta SOAP* ed è formato da un header contenente metadati e un body con informazioni sull'azione da eseguire e i parametri (o il risultato dell'azione nel caso della risposta).

SOAP è orientato alle azioni, le cihamate assumono nomi simili alle chiamate di procedure in Java (es. `getStudenti`, `updateStudente`).

Utilizza solo il metodo POST e pertanto non effettua caching delle richieste.

I vantaggi sono le estensioni di XML e la possibilità di validare la richiesta con il DTD. Lo svantaggio principale è la verbosità dei messaggi.

Un altro vantaggio di SOAP è il **Web Services Description Language** (WSDL): formato per la documentazione dell'API, insieme all' **Universal Description Discovery and Integration** (UDDI): standard per la costruzione di repository e motori di ricerca per web services.

API REST

REST è l'acronimo di **Representational State Transfer**

Caratteristiche

Si tratta di uno stile di programmazione di web api divenuto uno standard di fatto.

A differenza di SOAP l'idea di REST è di strutturare gli endpoint non con l'obiettivo di eseguire sottoprogrammi ma di accedere a risorse (tipicamente nel dbms).

Le api rest sono comunque basate su HTTP ma utilizzano come standard di serializzazione dati il formato JSON, molto più snello e orientato anch'esso agli oggetti.

URI

Gli endpoint sono organizzati come un filesystem virtuale. I nomi sono plurali se si riferiscono agli oggetti di una collezione, singolari se si riferiscono ad un oggetto singolo.

Esempi:

- `/studenti`
- `/esami`
- `/studenti/1234` studente con matricola 1234
- `/studenti/1234/esami`
- `/studenti/1234/esami/pp/insegnamento` insegnamento associato all'esame pp dello studente 1234

Gli URI possono riferirsi sia a proprietà e risorse reali che virtuali (ovvero calcolate dinamicamente).

Richieste

A differenza di SOAP (che utilizza solo il metodo POST), il protocollo REST utilizza tutti e 4 i protocolli principali (GET, POST, PUT, DELETE).

- Una richiesta di tipo GET serve per ottenere lo stato degli oggetti a cui ci si riferisce attraverso l'URI, il formato predefinito della risposta è la serializzazione dell'oggetto in JSON. Quando il server vuole restituire lo stato

dell'oggetto parzialmente (eliminando alcune proprietà o arricchendolo con proprietà calcolate) utilizza un DTO.
- Una richiesta POST aggiunge nuove risorse inviando nel corpo della richiesta la risorsa da aggiungere in formato json.
- In maniera simile lavora il metodo PUT procedendo all'aggiornamento della risorsa
- Il metodo DELETE elimina risorse esistenti

La query string nello standard REST viene utilizzata al solo scopo di filtrare il risultato di una richiesta GET, es: `GET /studenti?anno=2`.

I metodi PUT e DELETE sono idempotenti, ovvero esecuzioni ripetute danno gli stessi risultati, POST no, ogni ripetizione aggiunge nuovi dati clonati.

Operazioni non CRUD

Gli URI che fanno riferimento a esecuzioni di metodi sono caratterizzati da verbi e possono utilizzare sia GET che POST (più frequente).

Es. `POST /utenti/login`, `POST /conti/conto123/effettuaBonifico`

Convenzionalmente per separare versioni diverse della stessa API e mantenerle entrambe senza causare la rottura dei client che utilizzano la veccchia versione si antepone il nome di versione agli endpoint.

Es. `GET /v1/studenti` , `GET /v2/studenti`

Test e Documentazione

REST non ha uno standard per la documentazione. Esiste uno standard di fatto chiamato **OpenAPI** e basato su **Swagger**.

Swagger è uno strumento per generare documentazione a partire da un'API REST in json e fornisce un'interfaccia in HTML per l'interrogazione rapida delle API.

Gli strumenti di test degli endpoint sono essenziali durante lo sviluppo in quanto spesso lo sviluppo del/dei frontend non è parallelo a quello del backend. Un tool che permette di effettuare richieste alle API è postman.

Programmazione Web - J2EE

Lo sviluppo web in java lato server richiede **J2EE**, inizialmente Java 2 Enterprise Edition, dal 2017 Jakarta Enterprise Edition.

Server applicativo

Il server di riferimento per i backend in Java è Tomcat, progetto open source in ascolto di default sulla porta 8080. E' in grado di servire anche file statici

ma per una questione di prestazioni viene di solito utilizzato in accoppiata con un server HTTP standard. Quest'ultimo funzionerà da intermediario, quando gli sarà richiesta una risorsa statica la restituirà direttamente, quando la risorsa richiederà esecuzione del codice Java, il lavoro sarà passato al server tomcat.

Java Server Pages

Tecnologia attualmente poco utilizzata, è una pagina in cui sono mischiati codice HTML e codice Java tradotta automaticamente dal server in un servlet equivalente.

```
<html>
    <body>
        <%
            String nome = request.getParameter("nome");
            session.setAttribute("nome", nome);
        %>
        <p>Benvenuto <%= nome %></p>
    </body>
</html>
```

Struttura delle applicazioni

Una applicazione J2EE è una cartella che deve avere una struttura definita ed essere visibile sul filesystem virtuale.

La radice del filesysem virtuale corrisponde a una cartella chiamata *webapps*.

E' necessario che tutte le applicazioni seguano la struttura standard per essere correttamente supportate dal server tomcat.

Organizzazione dei file

- Cartella radice con pagine jsp, html, css e immagini
- Cartella **WEB-INF** contenente il file **web.xml**
- Cartella WEB-INF/classes contenente servlet e componenti
- cartella WEB-INF/lib con le librerie necessarie

La struttura della cartella viene solitamente generata dall'IDE nell'operazione di build. Convenzionalmente, anche se è possibile configurare il server per avere librerie condivise tra applicazioni, si preferisce incorporare con ogni applicazione una copia delle librerie necessarie in modod da evitare problemi con le versioni.

Tomcat ha una cartella `/lib` nella quale sono contenute le librerie comuni a tutti i progetti e le librerie `servlet-api.jar` e `jsp-api.jar`, non incluse in J2SE necessarie per lo sviluppo di applicazioni web.

WAR

I Web Application Archive sono archivi compressi con estensione .war e un'organizzazione di file e cartelle standard. I contenitori possono decomprimere e installare le applicazioni fornitegli sottoforma di file war.

Deployment Descriptor

Il file **web.xml** è un descrittore di parametri di configurazione. Attraverso questo file è possibile specificare parametri che occorrono al server per il deployment dell'applicazione.

Es. nomi e uri per servlet, timeout ecc...

Se è vuoto il contenitore assegnerà dei valori di default.

Gli elementi devono comparire nel seguente ordine:

- display-name
- description
- servlet
- servlet-mapping
- error-page

Ogni applicazione ha un nome (*context path*) che corrisponde al nome della cartella in cui è contenuta.

Deployment

Le applicazioni web per essere rese operative devono subire un processo di deployment. Questo differisce in base al server e all'applicazione, solitamente consiste nella copia dei file sul server, la configurazione di alcuni parametri, l'avvio di alcuni servizi. I parametri relativi alla configurazione vengono specificati nel file **Context Descriptor** di norma chiamato *context.xml*.

Lo standard di deployment per le applicazioni web su Tomcat prevede la copia del file .war nella cartella webapps. A questo punto il server riconosce la presenza del nuovo file e installa l'applicazione, in particolare decomprime il war in una cartella omonima, assegna un context path, utilizza il context descriptor per il deployment o ne crea uno standard.

Una applicazione può essere messa in stato di *stop* ovvero spenta o rimossa con l'azione *undeploy*. Il **reload** consente di reinizializzarne i componenti senza doverla rimuovere e reinstallare.

Per ovviare ai problemi di lentezza della procedura di deployment standard Tomcat offre una modalità alternativa che consiste nel creare un alias fornendo un descrittore di contesto in cui è indicata la posizione della cartella eseguibile da recuperare. # Servlet

Un servlet è una classe Java orientata alla comunicazione client server in grado di gestire richieste dai metodi principali.

Due package:

- `javax.servlet` per i servlet generici
- `javax.servlet.http` per i servlet http

L'interfaccia è `Servlet`, i servlet generici estendono la classe astratta `GenericServlet`, i servlet http estendono `HttpServlet`.

Il ciclo di vita dei servlet è gestito unicamente dal contenitore. Lo sviluppatore implementa metodi di servizio che saranno chiamati dal contenitore alla ricezione di una richiesta.

I metodi da implementare per gestire le principali richieste http sono `doGet()`, `doPost()`, `doPut()` e `doDelete()`. Le implementazioni ereditate sono vuote.

Tutti i metodi prendono come parametri un oggetto `HttpServletRequest` e un `HttpServletResponse`.

Ciclo di vita

Il ciclo di vita di un servlet prevede 3 fasi:

1. Inizializzazione (creazione delle istanze del servlet)
2. Servizio (istanze in funzione che gestiscono le richieste)
3. Distruzione (istanza distrutte dal garbage collector)

Il contenitore solitamente crea un'unica istana del servlet quando l'applicazione viene avviata ed esegue il metodo `init()`. Il servlet gestisce le richieste creando un thread per ogni nuova richiesta. Il thread chiama il metodo `service()` e il metodo service chiama il metodo necessario in base al tipo di richiesta.

Quando il server viene spento o viene spenta l'applicazione il contenitore chiama il metodo `destroy()` e distrugge l'istanza del servlet.

Richieste

Nella gestione delle richieste i servlet hanno a disposizione metodi per accedere ad alcuni parametri. Una richiesta contiene una mappa di parametri forniti nella querystring ai quali si può accedere con il metodo `String getParameter(String nomeParametro)` o `String[] getParameterValues(String nome)`.

La richiesta può anche accedere a:

- informazioni sul client
- informazioni sul metodo
- informazioni sull'URI
- informazioni sulle intestazioni http

Risposte

Un oggetto risposta consente di specificare le intestazioni http da inviare e il corpo del messaggio. Le intestazioni si impostano con i metodi `setHeader(String nome, String valore)` o `addHeader(String nome, String valore)`, l'intestazione fondamentale (e la prima da inserire) è il content type, impostabile con il metodo `response.setContentType("text/html")`.

Per scrivere il corpo della risposta è necessario ottenere un oggetto PrintWriter con il metodo `response.getWriter()` e successivamente scrivervi le righe del corpo.

Context

L'oggetto `javax.servlet.ServletContext` rappresenta il contesto nel quale il servlet viene eseguito, può essere recuperato con il metodo `getServletContext()`. Il context consente la comunicazione tra servlet e contenitore e consente di manipolare gli attributi dell'applicazione con il metodo `getAttribute(String chiave)`, `setAttribute(String chiave, Object obj)` e `removeAttribute(String chiave)`.

Il context risulta l'unica soluzione in caso di redirectm infatti i servlet chiedono al contenitore di effettuare l'inoltro della richiesta verso un altro servlet. Il compito è gestito dal **Request Dispatchet**, il cui riferimento può essere ottenuto con il metodo `getRequestDispatchet(String uri)` del ServletContext. Il dispatchet a sua volta ha un metodo `forward` che prende come parametri la richiesta e la ripsosta elaborata fino a quel momento e le inoltra al servlet di competenza.

Filtri

Un filtro è un servlet speciale che intercetta richieste e risposte da e verso altri servlet. Si utilizza un filtro per modificare una risposta prima di inviarla al client o per controllare determinate caratteristiche della richiesta e decidere se procedere con l'inoltro al servlet (tipico caso l'autorizzazione del client).

Anche i filtri devono essere definiti nel deployment descriptor e devono rispondere ad un url-pattern. Un url-pattern può essere associato a più filtri, e in questo caso questi verranno chiamati in cascata secondo l'ordine con il quale soo dichiarati nel file `web.xml`.

L'interfaccia da implementare per creare un filtro è `javax.servlet.Filter`, contenente 3 metodi: `init()`, `destroy()` e `doFilter()`, il terzo prende come parametri una richiesta, una risposta, e una `FilterChain`.

```
public class Filtro implements Filter {
    public void init(FilterConfig config) throws ServletException {}
    public void destroy() {}
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException, ServletException {
```

```
        chain.doFilter(reques, response); // necessario per inoltrare la richiesta
    }
}
```

Esempio dichiarazione di un filtro in *web.xml*:

```
<filter>
    <filter-name>FiltroAutorizzazione</filter-name>
    <filter-class>it.unibas.FiltroAutorizzazione</filter-class>
</filter>
<filter-mapping>
    <filter-name>FiltroAutorizzazione</filter-name>
    <url-pattern>*</url-patern>
</filter-mapping>
```

Security Context

La sicurezza in una webapp viene gestita nel package `javax.security.enterprise` di J2EE. Una classe di interfaccia `SecuriyContext` viene istanziata dal contenitore.

L'oggetto fornisce accesso al **principal** con il metodo `Principal getPrincipal()`

Principal

Il principal è un oggetto che consente di sapere se un utente è autenticato e risalire ai suoi permessi e ai suoi dati. Il metodo `String getName()` consente di conoscere il nome dell'utente autenticato e resituisce `null` in caso di mancata autenticazione.

Eventi dell'applicazione

Alcuni tipi di eventi che un'applicazione web potrebbe registrare:

- Creazione dell'applicazione
- Rimozione dell'applicazione
- Modifiche alla mappa dell'applicazione

I primi due sono intercettabili implementando l'interfaccia `javax.servlet.ServletContextListener` tramite i metodi `contextInitialized()` e `contextDestroyed()`. Per monitorare le modifiche alla mappa è necessario implementare l'interfaccia `javax.servlet.ServletContextAttributesListener` e utilizzare i metodi `attributeAdded()`, `attributeRemoved()`, `attributeReplaced()`.

Gli eventi vanno registrati nel deployment descriptor sotto il tag `<listener>` specificandone la `<listener-class>`. # Gestione delle date da Java 8

A partire da java 8 viene introdotto il package `java.time` che risolve alcuni problemi dovuti a scelte superate nella gestione delle date e semplifica il tutto prendendo spunto dalla piattaforma .NET.

Le classi per la gestione delle date sono `LocalDate`, `LocalTime` e `LocalDateTime`.

`LocalDate` permette di creare una data con il metodo statico `of(<anno>, <mese>, <giorno>)`. A differenza della gestione con Calenadar la numerazione dei mesi è a base 1 e il comportamento di default non è permissivo: il costruttore solleva `java.time.DateTimeException`. Per ottenere la data attuale: `LocalDate.now()`.

`LocalTime` offre il metodo statico `of(<ore>,<minuti>,<secondi>)`, supera il limite di Calendar di non poter rappresentare solo orari.

Tutti gli oggetti del package hanno un metodo `format(<formatter>)` che prende come parametro un oggetto della classe `java.time.format.DateTimeFormatter`.

Per costruirlo è necessario specificare pattern e localizzazione: `DateTimeFormatter.of(<pattern>).withLocale`

Il pattern però non è localizzato, l'alternativa è richiedere la stampa localizzata con il metodo `ofLocalizedDate` che accetta un `FormatStyle` che può assumere come valor `FULL`, `LONG`, `MEDIUM` e `SHORT`.

```
LocalDate d = LocalDate.now();
DateTimeFormatter formatter = DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM).withLoca
System.out.println("Data odierna: " + d.format(formatter));
```

Un `LocalDateTIme` è composto da un LocalDate e un LocalTime: `LocalDateTime.of(<localDate>, <localTime>)`.

Per accedere a parti della data mette a disposizione metodi predefiniti:

- `getDayOfMonth()`
- `getMonth()`
- `getYear()`
- `getHour()`
- `getMinute()`
- `getSecond()`

Tutte le classi del package offrono metodi per aggiungere o sottrarre intervalli di tempo:

- `plusMonths()`
- `plusDays()`
- `plusWeeks()`
- `plusYears()`
- ...

Lombok

Uno degli svantaggi di Java è l'eccessiva verbosità. Project Lombok è una libreria opensource nata per snellire lascrittura di codice Java.

Fornisce una serie di annotazioni che verranno elaborate dal preprocessore che le sostituirà con parti di codice prima di sottoporre la classe al compilatore.

Ad esempio l'annotazione `@Getter` inserisce implicitamente tutti i metodi getter per le proprietà della classe.

Per questa ragione è fondamentale il supporto degli IDE che devono gestire l'autocompletamento di metodi che non esistono (perchè implicitamente dichiarati nelle annotazioni lombok). Attualmente gli IDE principali lo supportano.

Annotazioni

- @Getter
- @Setter
- @ToString
- @EqualsAndHashCode (sulla base di tutte le proprietà, di default)
- @NoArgsConstructor
- @AllArgsConstructor
- @RequiredArgsConstructor (solo le proprietà final)
- @Slf4j (crea un oggetto logger `log`)
- @Data (ToString, EqualsHashCode, Getter, Setter e RequiredArgsConstructor)
- @Value (oggetti immutabili con proprietà tutte final, ToString, Equals..., Getter, RequiredArgsConstructor)
- @NotNull (genera una guardia se la proprietà diventa null solleva NullPointerException)

Proprietà delle annotazioni

Le annotazioni hanno proprietà che controllano il funzionamento e vengono inserite nel costruttore dell'annotazione. Es.`@ToString(callSuper=[call | skip| warn], of = {lista-dei-campi}, ...)`

Sulle proprietà della classe è possibile inserire annotazioni per includerle/escluderle es. `@ToString.Include` o `@EqualsHashCode.Exclude`.

Pattern Builder

Il **Builder** è un design pattern che risolve il problema della creazione di un oggetti con molte proprietà di cui alcune hanno valori di default.

Una soluzione potrebbe essere creare un costruttore vuoto e implementare tutti i metodi setter per le proprietà. Soluzione verbosa.

Un'altra è creare molti costruttori con argomenti uno per ogni variante di argomenti non nulli. Anche questa complessa e verbosa.

Il pattern builder risolve il problema tramite un oggetto esterno con un metodo

`build()` e procedure per impostare le proprietà utilizzabili con uno stile conversazionale.

Dichiarazione

```
@Builder
public class Utente {
    private String nomeUtente;
    private String password;
    private String nome;
    @Builder.Default
    private String dipartimento = "Staff";
    @Builder.Default
    private String ruolo = "Utente";
}
```

Utilizzo

```
Utente u1 = Utente.builder()
                .nomeUtente("u1@email.it")
                .password("passowrd")
                .nome("Utente uno")
                .build();
Utente u2 = Utente.builder()
                .nomeUtente("u2@email.it")
                .password("password2")
                .nome("Utente due")
                .dipartimento("IT")
                .ruolo("Admin")
                .build();
```

JAX-RS

Un framework per la creazione di applicazioni web J2EE fornisce uno scheletro di applicazione web con un servlet principale che intercetta tutte le richieste e gestisce/genera tutti i componenti necessari sulla base di annotazioni scritte dal programmatore.

E' fondamentale il concetto di **bean gestito**: il programmatore predispone il codice ma sarà il framework a istanziare le risorse necessarie e chiamare i metodi che il programmatore ha definito sulla base di quello che avviene nell'applicazione.

I principali framework per lo sviluppo backend in Java sono Jersey, REST Easy, RESTlet e Spring Boot. Dei quali il primo e l'ultimo occupano la maggior fetta di mercato.

I framework forniscono solitamente anche servizi per la validazione dei bean e per la dependency injection.

JAX-RS, acronimo inizialmente di **Java API for RESTful Web Services**, ora **Jakarta RESTful Web Services** è uno standard per i framework orientati allo sviluppo di API REST. Definisce le interfacce dei componenti e le annotazioni standard da utilizzare.

Il framework Jersey è l'implementazione di riferimento dello standard.

Elementi di un'app JAX-RS

- Resources
- Services
- Providers
- Modello
- DAO
- DTO

Resource

Le resource sono le classi responsabili della gestione di richieste e risposte. Ogni classe resource gestisce le richieste per una risorsa, ogni metodo gestisce un endpoint.

Annotazioni principali per le risorse ___@Path___: può essere applicata alla classe e al metodo e associa un URI alla risorsa (classe) o un URI di un endpoint al metodo che lo gestisce. Es. `@Path("/studenti")` sulla classe, `@Path("{idStudente}/mediaPesata")` sul metodo.

@GET___, @POST___, **@PUT___,** @DELETE___: associano il metodo HTTP al metodo che gestisce un endpoint

@Produces(MediaType): consente di specificare il formato della serializzazione, il formato standard per REST è `MediaType.JSON`.

@Consumes(MediaType): analogo del precedente per le richieste con corpo.

Binding dei parametri Per recuperare parti dell'URI o della query string o intestazioni passate con la richiesta esistono annoazioni specifiche:

@PathParam("id"): prende il parametro `{id}` nel Path. (es. /studenti/{id}/media)

@QueryParam("id"): prende il valore di *id* passato nella querystring

@HeaderParam("id"): prende il valore dell'intestazione *id*

@CookieParam("id"): prende il valore del cookie con nome *id*

Esempio di risorsa CRUD

```java
@Path("/studenti") @Produces(MediaType.APPLICATION_JSON)
public class RisorsaStudenti {
    @GET
    public List<StudenteDTO> getAll() {...}
    @GET @Path("/{id}")
    public StudenteDTO get(@PathParam("id") long id) {...}
    @POST @Consumes(MediaType.APPLICATION_JSON)
    public long add(StudenteDTO s) {...}
    @DELETE @Path("/{id}")
    public void delete(@PathParam("id") long id) {...}
}
```

Modello e DAO e DTO sono componenti ordinari.

Services

Sono operatori utilizzati dalle resources per costruire le risposte. Le Resource infatti sono convenzionalmente senza stato e si occupano solo di gestire richieste e produrre risposte. La logica applicativa si trova nei services che sono perciò modulari e riutilizzabili.

L'operatore ___@Provider___ ha lo scopo di selezionare un componente in modo che sia *scoperto* dal framework nell'analisi del classpath. Un utilizzo classico sono le classi per i filtri.

Dependency Injection

La dependency injection è una alternativa alla **service location** per applicazioni in cui la quantità di componenti renda difficile quest'ultima.

Si chiamerà *client* il componente che intende acquisire riferimenti di un altro componente, chiamato *server* (nulla a che vedere con architetture client-server di rete).

La dependency injection prevede che il componente client debba soltanto dichiararli e utilizzarli senza localizzarli esplicitamente. Sarà un terzo componente esterno, detto **contenitore** o **contesto** a localizzare, creare e *iniettare* i componenti richiesti dal client.

Tutti i bean oggetto di injection devono essere gestiti dal framework che deve controllarne il ciclo di vita.

La tecnica prende il nome di **inversione del controllo** e il fenomeno va sotto il nome di **Hollywood principle** (il framework dirà al componente "don't call me, I will call you").

In questo modo il componente ha codice molto più pulito in quanto dichiara semplicemente i componenti che intende utilizzare e sarà poi compito del framework acquisirli.

Il framework non può iniettare bean che non gestisce

Scope di un bean gestito

Lo scope è l'ambito di visibilità e il ciclo di vita di un bean, stabilisce quindi la politica di creazione e iniezione. 2 scope:

1. Application scope: unica istanza per tutta l'applicazione (simile al concetto di singleton)
2. Request scope: un'istanza del bean per ogni richiesta

Lo standard per le DI è pensato per essere utilizzato in qualsiasi tipologia di applicazione. Uno standard specifico **Context Dependency Injection** include alcune annotazioni specifiche per la programmazione web.

Jakarta Context Dependency Injection

E' l'implementazione di riferimento dello standard controllata da Eclipse.

Per indicare al framework i punti di iniezione si utilizza l'annotazione ___@Inject___ applicata a proprietà, metodi o costruttori.

es `@Inject ServiceStudenti serviceStudenti`

Per gli scope esistono le annoazioni `@ApplicationScoped` o `@RequestScoped` da applicare ai bean gestiti.

L'annotazione `@Context` viene utilizzata per iniettare in una risorsa il riferimento ad un elemento collegato al contesto dell'applicazione web. Caso tipico è l'iniezione del security context: `@Context SecurityContext context` utilizzato per risalire al Principal e recuperarne l'identità.

Elementi iniettabili:

- SecurityContext
- ServletContext
- HttpServletRequest
- HttpServletResponse
- HttpHeaders
- UriInfo
- ServletConfig
- Providers, Configuration...

Bean validation

Lo standard per la validazione dei valori in Java è **Jakarta Bean Validation**, Hibernate Validation è l'implementazione di riferimento.

Anche in questo caso il framework fornisce una serie di annotazioni per definire vincoli sui valori delle proprietà. La convalida può essere innescata manualmente o automaticamente dal framework che solleva eccezioni in caso di violazioni.

Principali annotazioni:

- ___@NotNull___
- ___@NotEmpty___ (stringa o collezione)
- ___@NotBlank___ (stringa)
- ___@Positive, @PositiveOrZero___ (numero)
- ___@Negative, @NegativeOrZero___ (numero)
- **@Min(value=m)** (numero)
- **@Max(value=m)** (numero)
- ___@AssertTrue___ (boolean)
- **@Size(min=m, max=M)** (stringa o collezione)
- ___@Email___
- ___@Past, @PastOrPresent___ (date)
- ___@Future, @FutureOrPresent___ (date)

Le annotazioni possono essere posizionate in corrispondenza di proprietà o parametri dei metodi ed è possibile er ognuna aggiungere un messaggio di errore.

```
@NotNull(message="Non deve essere vuoto") String nome;
```

L'annotazione ___@Valid___ associata ad un riferimento ad un bean con vincoli di validità innesca la validazione dei vincoli nella classe (richiede che il bean sia valido).