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)