APPUNTI TIPSIT (PIETRO VACCARI)

Crittografia (Teoria)

E' la scienza che studia come rendere segreta e sicura la comunicazione tra due persone o entità nascondendo il significato del messaggi.

Con questo termine si intende oggi un insieme di tecniche che consentono di trasmettere messaggi mantenendoli segreti a tutti, tranne ad alcune persone che possiedano la chiave per comprenderli.

Definizioni

Segretezza: il messaggio non deve essere leggibile a terzi.

Autenticazione: il destinatario deve poter essere sicuro del mittente.

Affidabilità dei documenti: si intende di avere la garanzia e la certezza che un documento sia originale, cioè che il suo mittente sia certo (ad esempio mediante l'apposizione su di esso di una firma digitale) e che non sia stato letto e/o alterato e modificato da altre persone non autorizzate.

Integrità: il destinatario deve poter essere sicuro che il messaggio non sia stato modificato.

Attendibilità: il mittente non deve poter negare di aver inviato il messaggio.

Cifratura: è l'operazione con la quale si nascondono le informazioni; essa viene effettuata tramite un procedimento chiamato cifrario.

Testo in chiaro: è il messaggio da cifrare.

Testo cifrato: è il messaggio trasformato in modo da non essere più leggibile tramite una semplice lettura.

Decifrazione: è la riconversione di un testo cifrato nella sua forma originaria, cioè nel testo in chiaro.

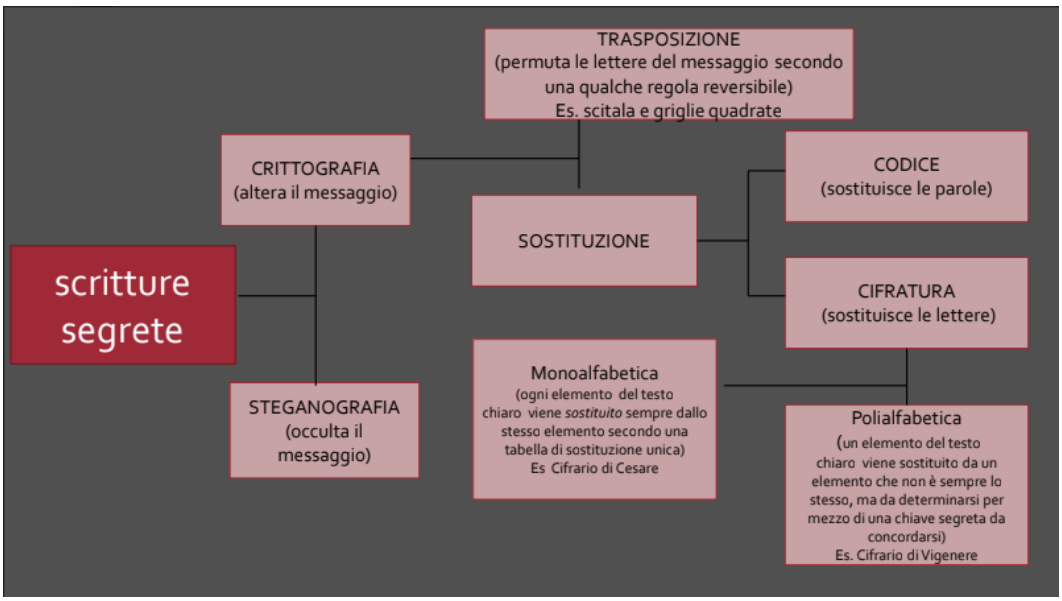
Cifrario: è il procedimento (algoritmo) che consente di crittare e decrittare i testi.

Il processo di trasformazione dal messaggio in chiaro al messaggio cifrato e viceversa è spesso noto, ma si basa su una informazione specifica (detta "chiave"), senza la quale non si è in grado di operare.

Utilizzi moderni della crittografia

- L'uso più importante della crittografia in ambito "civile" è quella della sicurezza delle comunicazioni in rete.
- Più in particolare le applicazioni di commercio elettronico sono quelle in cui maggiormente è sentita la necessità della sicurezza e della segretezza (scambio di dati sensibili, quali il numero di carta di credito, numero di conti bancari, ecc.)
- Un altro utilizzo importante è quello della firma digitale e dell'autenticazione dei

documenti, che ha applicazioni nella pubblica amministrazione (e-government) e in generale negli aspetti burocratici (contratti, domande, moduli, vari documenti ufficiali, ecc.)



Sostituzione monoalfabetica

- Il caso più generale è quello in cui l'alfabeto cifrato è una permutazione di quello in chiaro.
- La chiave di cifratura è la lista delle cifre in ordine alfabetico; se si cifra con lettere dello stesso alfabeto sarà un alfabeto disordinato impossibile da ricordare a memoria.
- Un modo semplice per ricordare la permutazione è quello di usare una frase (chiave) le cui lettere fanno da inizio della lista (escludendo quelle ripetute), lasciando poi le altre lettere a seguire.

Sicurezza:

Esistono 21!=51090942171709440000 permutazioni possibili, cioè circa 51 · 1018, ossia più di cinquanta miliardi di miliardi.

Una ricerca esaustiva per trovare la permutazione giusta è praticamente impossibile , eppure questo codice è tutt'altro che sicuro...

Usare simboli diversi al posto delle lettere non aumenta la difficoltà di decifratura.

Nella crittoanalisi, l'analisi delle frequenze è lo studio della frequenza di utilizzo delle lettere o gruppi di lettere in un testo cifrato. Questo metodo è utilizzato per violare i cifrari classici.

In particolare un primo metodo che si adotta in attività di crittanalisi si basa sul fatto che in ogni lingua la frequenza di uso di ogni lettera è piuttosto determinata; questo è vero in modo rigoroso solo per testi lunghi, ma spesso testi anche corti hanno frequenze non molto diverse da quelle previste.

Sostituzione polialfabetica

- Un crittosistema per sostituzione polialfabetica non usa sempre lo stesso simbolo per la stessa lettera.
- Per ogni posizione del messaggio in chiaro viene usato un alfabeto diverso.
- Così si supera (parzialmente...) il problema della debolezza visto per i crittosistemi monoalfabetici.
- Infatti ha "meno" senso contare le frequenze di simboli, non essendoci piùcorrispondenza tra lettere in chiaro e simboli cifrati.

Le cifrature per trasposizione

- I sistemi di trasposizione letterale consistono nel rimescolare i caratteri del testo chiaro secondo una qualche regola reversibile.
- Le trasposizioni più semplici consistono nell'invertire il testo intero o le singole parole, soluzioni ovviamente debolissime dal punto di vista della segretezza.
- Sistemi più impegnativi richiedono di disporre il testo su una matrice rettangolare, righe e colonne; la trasposizione più semplice consiste allora nel disporre il testo su n righe e quindi scambiare righe con colonne.
- Un po' più sofisticata l'idea di rimescolare le colonne in base a una parola chiave segreta trasposizione con chiave) o usando un qualche dispositivo come le griglie.

- L'uso dei sistemi a trasposizione declinò bruscamente con la nascita delle prime macchine cifranti, quasi tutte basate su sistemi di sostituzione letterale. Alla fine del XX secolo con l'avvento dei computer la trasposizione ha conosciuto una seconda giovinezza, a livello informatico è infatti facile realizzare sistemi di trasposizione anche molto complessi; oggi la trasposizione è usata soprattutto in combinazione (sovracifratura) con la sostituzione come nei cifrari DES e AES, dove però la trasposizione è a livello di blocchi di bit non più di lettere alfabetiche.

  Nei sistemi di trasposizione letterale il testo cifrato è un anagramma del testo originale; matematicamente si parla di una permutazione, e il numero di permutazioni possibili è dato dal fattoriale: $Pn = n! = n \times (n-1) \times (n-2)...1$ In realtà questa è una stima per eccesso, valida solo se le n lettere sono tutte distinte, mentre è più probabile, quanto più lungo è il testo, che vi siano lettere ripetute; in tal caso le permutazioni possibili sono molte di meno; in generale vale la formula, per un testo di n lettere, con m lettere ripetute, la prima n1 volte, la seconda n2 volte, ecc.: $Pn = n!/(n1! \times n2!...)$

Le griglie quadrate

Le griglie sono uno dei sistemi per trasposizione più antichi e semplici; consistono di un cartoncino o lamina metallica o di legno, con una serie di fori. Il messaggio chiaro viene scritto nei fori secondo una qualche regola e il cifrato si ottiene leggendo i caratteri secondo una diversa regola, oppure muovendo la griglia secondo un ordine stabilito. Per decifrare si segue il procedimento inverso.

Le griglie quadrate di rotazione

- I fori devono essere la metà (griglia a due rotazioni) oppure un quarto (griglia a quattro rotazioni) del numero totale di caselle disponibili e devono essere disposti in modo da coprire tutte le caselle del quadrato una e una sola volta in quattro successive rotazioni.
- Per cifrare si dispone la griglia nella posizione iniziale sopra un foglio di carta, e si scrive il messaggio nei fori disponibili fino ad esaurimento, quindi si ruota la griglia di 180° (due rotazioni) o 90° (quattro rotazioni) e si continua fino ad esaurimento del messaggio; se il messaggio è più breve del numero totale di caselle si possono riempire le caselle avanzate con nulle; se è più lungo si può ricominciare dalla posizione iniziale, oppure rovesciare la griglia...

- Più sicure e più usate sono le griglie a quattro rotazioni
- La griglia deve essere vista come formata da quattro sottogriglie di ordine dimezzato, destinate a sovrapporsi l'un l'altra secondo 4 rotazioni di un angolo retto; queste sottogriglie devono essere formate da un numero divisibile per 4, se si vogliono distribuire uniformemente i fori; altrimenti qualsiasi quadrato di ordine pari è accettabile.

Il cifrario di Vigenère

Come si è detto questo cifrario ha goduto per tre secoli la fama di essere un cifrario inattaccabile; nel 1863 il colonnello prussiano Friedrich Kasiski pubblicò un primo metodo di decrittazione. In realtà Anche se questa cifratura è molto più sicura di quella di Cesare, è anch'essa facilmente crackabile.

La debolezza del Vigenère sta nell'essere, di fatto, un insieme di n cifrari di Cesare, dove n è la lunghezza della chiave; se il crittoanalista riesce a determinare la lunghezza della chiave (nel nostro caso, n) la decrittazione diventa molto semplice.

Per far ciò si possono utilizzare metodi statistici per trovare n, e successivamente applicare l'analisi delle frequenze a ciascun alfabeto cifrante. La parte più complicata sta dunque nello scoprire la lunghezza della chiave cifrante anche se la cosa non è impossibile. Nel testo cifrato infatti, se la chiave utilizzata è breve, vi saranno probabilmente delle serie di lettere ripetute.

Queste serie di lettere, se abbastanza lunghe (5-6 caratteri), saranno generate probabilmente dalla stessa parola in chiaro.

Basterà allora calcolare la distanza tra una parola e l'altra e la lunghezza della ripetizione per risalire alla lunghezza n della chiave. Così facendo si può capire quali lettere usano il primo alfabeto ifrante, quali il secondo, e così via procedendo poi con l'analisi delle frequenze per ciascun alfabeto.

Per evitare questo problema, una soluzione consiste nell'usare una chiave di dimensioni simili a quella del testo per rendere impossibile uno studio statistico del testo criptato. Questo tipo di sistema di cifratura è detto Sistema a chiave usa e butta. Il problema di questo metodo è la lunghezza della chiave di cifratura (più il testo da criptare è lungo, più la chiave deve essere voluminosa), che impedisce la sua memorizzazione e include una probabilità d'errore nella chiave maggiore (un solo errore rende il testo indecifrabile, ecc.).

- I cifrari tradizionali sono oggi detti a chiave segreta in quanto richiedono che i due comunicanti debbano preventivamente concordare una chiave segreta.
- A differenza dei cifrari tradizionali, basati su una chiave segreta, questi cifrari usano anche una chiave pubblica, garantita da una qualche autorità.

Le misure da intraprendere per ottenere la segretezza possono essere anche affrontate in diversi livelli della pila protocollare: a livello fisico si può cercare di impedire che avvengano intercettazioni di dati, a livello di data link si possono introdurre codifiche dei dati trasmessi per renderli incomprensibili agli hacker.

È comunque il livello di applicazione che può gestire gli altri due problemi ed è su quello che noi concentreremo la nostra attenzione.

Classe dei resti

Dato un numero intero positivo X, i numeri interi si distribuiscono in X classi di resto modulo m, a seconda del resto che danno quando vengono divisi per m.

Valgono inoltre le seguenti due equivalenze:

$(X+Y)(mod m) = X(mod m) + Y(mod m)$, e cioè: il resto di una somma è pari alla somma dei resti

$(XY)(mod m) = X(mod m)Y(mod m)$, e cioè: il resto di un prodotto è pari al prodotto dei resti.

L'equivalenza sul prodotto conduce alla importante equivalenza sul quadrato: il resto di un quadrato è pari al quadrato del resto

$$X^2(mod m) = (X \cdot X)(mod m) = x(mod m) \cdot x(mod m) = R \cdot R = R^2$$

Grazie a questa equivalenza sarà possibile determinare resti di divisioni fra numeri con un incalcolabile numero di cifre, base della crittografia a chiave pubblica che utilizza i numeri primi.

A $13^2 (\text{mod } 11) = 169 (\text{mod } 11) = 4 = 13 (\text{mod } 11) \cdot 13 (\text{mod } 11) = 2 \cdot 2 = 4$
B $25^2 (\text{mod } 7) = 625 (\text{mod } 7) = 2 = 25 (\text{mod } 7) \cdot 25 (\text{mod } 7) = 4 \cdot 4 = 16$

16, essendo maggiore di m, deve essere ulteriormente elaborato ottenendo:

$$16 (\text{mod } 7) = 2$$

Numeri primi

I numeri primi sono stati oggetto di studio dai matematici di ogni periodo storico: tutti sanno che un numero primo non è rappresentabile come prodotto di interi che lo precedono e si dice primo se è divisibile esattamente solo per 1 e per se stesso.

Teoremi di Euclide sui numeri primi

- Primo Teorema di Euclide: ogni numero intero N si scrive in modo unico (a parte l'ordine) come prodotto di numeri primi.
- Secondo Teorema di Euclide: i numeri primi formano una successione infinita.

Aritmetica modulare (calcolo del mod m)

Nella aritmetica modulare il quoziente nell'operazione di divisione è irrilevante mentre unica importanza lo assume il resto, e viene così indicato:

Q il quoziente della divisione fra il dividendo X e il divisore m, mentre è R il resto e viene indicato con la seguente notazione: $X(modm) = R$ che si legge: "X modulo m è uguale a R" e si dice anche "R è congruo a X modulo m".

- $14 (\text{mod } 4) = 2$
- $79 (\text{mod } 7) = 2$
- $21 (\text{mod } 33) = 21$ dalla quale deduciamo che $X (\text{mod } m) = X$ se $X < m$
- $37 (\text{mod } 37) = 0$ quindi $m (\text{mod } m) = 0$
- $27 (\text{mod } 1) = 0$ quindi $X (\text{mod } 1) = 0$
- $77 (\text{mod } 76) = 1$ quindi $(m + 1) (\text{mod } m) = 1$

💡 Esempio: **27(mod 4)** → 27/4 = 6.75 → 6*4 = 24 → 27- 24 = **3**

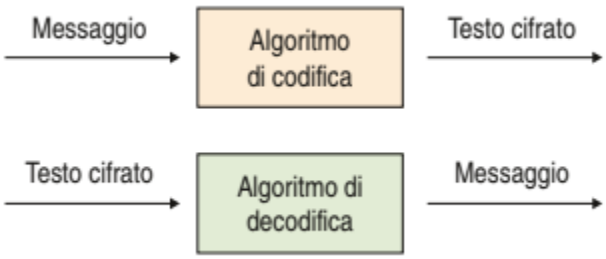
Quindi 27(mod4) = 3

Possiamo fare tre osservazioni sul resto R:

- vale sempre la relazione R < m;
- tutti i possibili resti sono in numero pari a m e con valori compresi fra 0 e m – 1, e l'insieme dei resti viene indicato con Zn = { 0, 1, 2, ..., n – 1};
- se X < m allora X(mod m)=X.

Tipi di Crittografia

Naturalmente le regole di cifratura devono essere note sia al mittente del messaggio che al destinatario, in modo che quest'ultimo possa, alla sua ricezione, effettuarne il decriptaggio e comprenderne il significato.

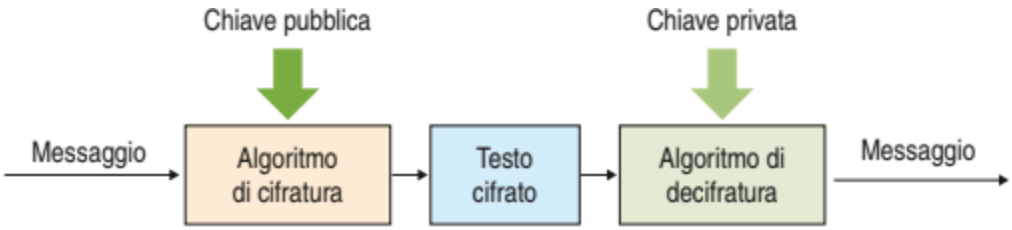


La regola di cifratura è generalmente composta da due elementi:

- La regola vera e propria (l'algoritmo utilizzato), che in questo caso consiste nella "sostituzione di un carattere con un altro";
- Uno (o più) parametri, in questo caso la posizione del carattere da prendere (nell'esempio il successivo a quello in chiaro: se invece si fosse trasmesso CMZVQ le posizioni sarebbero state due in avanti).

Quando la chiave di cifratura coincide con quella di decifratura lo schema crittografico si dice simmetrico e la chiave prende il nome di chiave comune.

Quando la chiave di cifratura è invece diversa da quella usata per la decifratura lo schema crittografico si dice asimmetrico e le due chiavi si chiamano chiave pubblica quella usata per la cifratura, che è comune a tutti i mittenti e di pubblico dominio, e chiave privata quella utilizzata per la decifratura, che è segreta e di conoscenza solo del destinatario del messaggio.



Questo procedimento è alla base della moderna sicurezza delle reti: il mittente non deve comunicare col destinatario o accordarsi preventivamente, ma utilizza la chiave pubblica del destinatario che, proprio perché pubblica, è a disposizione di tutti, e con essa prepara il messaggio da trasmettere criptandolo in modo tale che solo chi è in possesso della chiave privata lo può decriptare.

Crittoanalisi

Generalmente l'algoritmo di cifratura è noto e standardizzato, quindi conosciuto e soggetto a crittoanalisi da parte dei malintenzionati per individuare la chiave utilizzata e quindi decriptare il messaggio

- è l'arte della "rottura" dei codici e dei cifrari.
- La crittoanalisi studia come decifrare un messaggio senza esserne "autorizzati".
- La decrittazione è la riconversione di un testo cifrato nella sua forma originaria, cioè nel testo in chiaro , senza essere in possesso della chiave.
- E' lo studio dei metodi perottenere il signifi cato di informazioni cifrate: tipicamente si tratta delle operazioni effettuate alla ricerca della chiave segreta.

- La crittoanalisi ha il ruolo fondamentale di far capire quanto un sistema di cifratura/decifratura sia sicuro.

Principio di KERCKHOFFS

La sicurezza di un crittosistema deve dipendere solo dalla segretezza della chiave e non dalla segretezza dell'algoritmo usato.

La crittografia simmetrica, cifrari a sostituzione e a trasposizione

La crittografia simmetrica è un tipo di crittografia che utilizza una sola chiave per cifrare e decifrare i messaggi. In questo tipo di crittografia, la stessa chiave è utilizzata sia dal mittente che dal destinatario per cifrare e decifrare il messaggio.

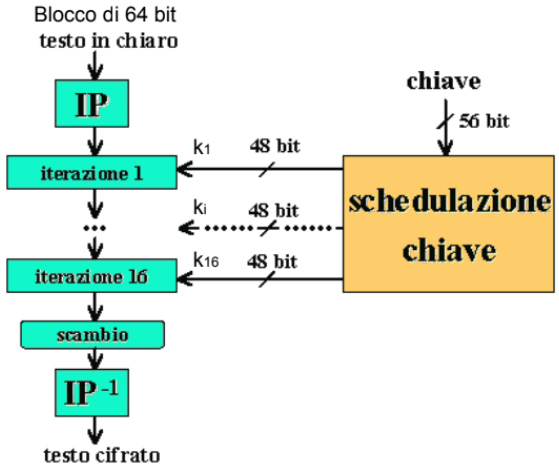
I cifrari a sostituzione sono un tipo di crittografia simmetrica che sostituiscono ogni carattere del messaggio originale con un altro carattere o simbolo. In genere, la sostituzione avviene in base ad una tabella di sostituzione predefinita, detta alfabeto segreto. Un esempio di cifrario a sostituzione è il cifrario di Cesare, che sostituisce ogni lettera del messaggio originale con la lettera spostata di un certo numero di posizioni nell'alfabeto.

I cifrari a trasposizione, invece, sono un tipo di crittografia simmetrica che rimescolano le lettere del messaggio originale. In questo tipo di cifrario, l'ordine delle lettere del messaggio viene cambiato in base ad una chiave segreta, detta chiave di trasposizione. Un esempio di cifrario a trasposizione è il cifrario di Colonna, che organizza le lettere del messaggio originale in colonne sulla base della chiave di trasposizione e poi le legge in righe.

In entrambi i casi, la crittografia simmetrica e i cifrari a sostituzione e a trasposizione sono tecniche che possono essere utilizzate per proteggere la privacy dei messaggi scambiati tra due parti. Tuttavia, questi metodi di crittografia possono essere vulnerabili ad attacchi di tipo brute-force o attacchi di crittanalisi, per cui è necessario utilizzare tecniche più avanzate di crittografia come la crittografia asimmetrica.

DES (data Encyption Standard)

Storia



1. 1973: Il National Bureau of Standards (NBS) pubblica un bando in cui richiede un algoritmo di cifratura:
    - La cui sicurezza risiedesse nella segretezza della chiave e non nel processo di cifratura
    - Che potesse essere realizzato efficientemente a livello hardware
2. IBM propone una prima versione del DES
3. NSA (National Security Agency) lo certifica ma propone delle variazioni:
    - Riduzione della lunghezza della chiave, da 128 bit a 56 bit
    - Modifica delle funzioni contenute nelle S-box
4. 1977: DES viene accettato e reso pubblicamente disponibile.
    - Primo cifrario certificato ufficialmente come sicuro e noto a tutti.
    - Certificato ufficialmente ogni 5 anni
5. 1987: Prime manifestazioni di sfiducia sul DES
6. 1998: Il DES viene rotto
7. 2000: NBS sceglie il successore del DES, denominato Advanced Encryption Standard (AES).

Le basi del DES

Proprietà desiderabili di un algoritmo di cifratura secondo Shannon.

- Diffusione: Alterare la struttura del testo in chiaro “spargendo” i caratteri su tutto il testo cifrato.
- Confusione: Combinare in modo complesso il messaggio e la chiave, per non permettere al crittoanalista di separare le due sequenze mediante l’analisi del

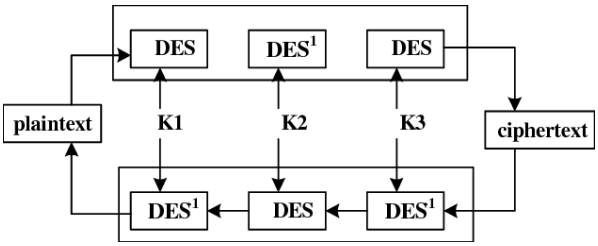
crittogramma.

La struttura del DES

- Un blocco di testo in chiaro (64 bit) viene permutato dal blocco IP, il risultato di questa permutazione andrà in ingresso alla prima iterazione.
- In ogni iterazione il blocco di 64 bit viene opportunamente mescolato con una Kiesima chiave.
- In tutto 16 iterazioni.
- Dopo le 16 iterazioni al testo di 64 bit oramai cifrato viene applicata la permutazione iniziale inversa IP^-1

3DES

Nel 1999 è stato introdotto il Triple DES, con 3 passaggi di cifratura DES consecutivi con 3 chiavi diverse per un totale di 168 bit: la maggiore sicurezza rispetto a quella del DES è proprio nella lunghezza tripla della chiave.



La crittografia asimmetrica (RSA), sicurezza nell'utilizzo delle chiavi

La crittografia asimmetrica, conosciuta anche come crittografia a chiave privata, è un sistema di crittografia che utilizza due chiavi distinte anziché una sola. Una delle chiavi è nota come chiave pubblica e può essere diffusa liberamente, mentre l'altra è nota come chiave privata e deve essere mantenuta segreta dall'utente.

L'algoritmo di crittografia più comune utilizzato nella crittografia asimmetrica è RSA, che prende il nome dai suoi tre creatori: Ron Rivest, Adi Shamir e Leonard Adleman. L'RSA è stato sviluppato negli anni '70, ma è ancora ampiamente utilizzato oggi per proteggere le comunicazioni su Internet.

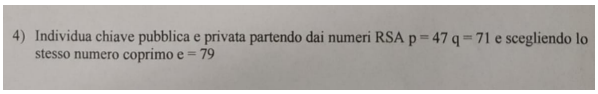
La sicurezza nella crittografia asimmetrica dipende dalla sicurezza delle chiavi. La chiave privata deve essere protetta con cura, poiché se caduta nelle mani sbagliate, può essere utilizzata per decifrare i messaggi crittografati con la chiave pubblica. Inoltre, la chiave pubblica deve essere autenticata in modo che l'utente possa essere sicuro di aver ricevuto la chiave pubblica corretta e non una chiave fraudolenta.

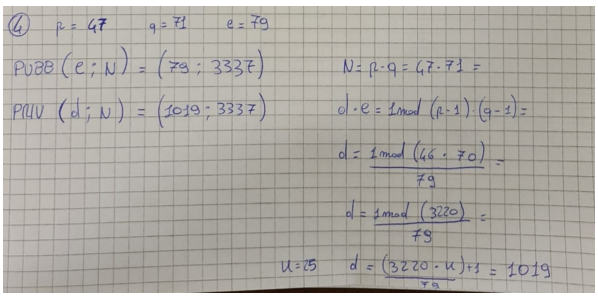
Per garantire la sicurezza nella crittografia asimmetrica, sono stati sviluppati molti protocolli di sicurezza, tra cui il protocollo SSL/TLS utilizzato per proteggere le comunicazioni su Internet. Questi protocolli utilizzano la crittografia asimmetrica per stabilire una connessione sicura tra i due utenti, e poi passano a un sistema di crittografia simmetrica per il resto della comunicazione.

In conclusione, la crittografia asimmetrica è un importante strumento per proteggere le comunicazioni su Internet, ma la sicurezza dipende dalla protezione delle chiavi e dall'adozione di protocolli di sicurezza adeguati.

Calcolo delle chiavi RSA

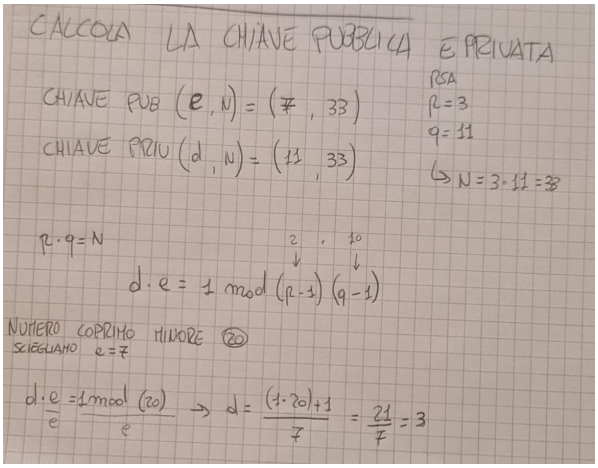
Vediamo come funziona RSA con due persone, A e B, che vogliono scambiarsi i messaggi. A è l'utente che genera due chiavi pubbliche (utilizziamo per l'esempio numeri primi piccoli).





- A genera due numeri primi distinti p e q e li moltiplica tra di loro ottenendo il numero N che viene reso pubblico, mentre p e q devono restare segreti.
  Esempio:
  $p = 5; q = 11$
  $p * q = 5 * 11 = 55$
  $N = 55$, prima chiave pubblica
- A calcola b (funzione di Eulero):
  $b = \Phi(n) = (p - 1) * (q - 1)$. Il numero b deve restare segreto.
  Esempio:

Φ(55) = (5 - 1) * (11 - 1) = 4 * 10 = 40
b = 40



- A calcola il primo intero e che sia primo con b (non abbia divisori in comune, ovvero $MCD(e, b) = 1$). Il numero e è la seconda chiave pubblica.
  Esempio:

  $e = 3$ perchè $MCD(3, 40) = 1$
  $e = 3$ seconda chiave pubblica

- A calcola il numero d inverso di e nella classe di congruenza modulo b, che è il più piccolo x per cui sia $e * dmod(b) = 1$; il numero d é la chiave per decifrare e deve restare segreta, chiave privata. Per il calcolo di d è efficiente un'estensione del classico algoritmo di Euclide per l'MCD (detto anche teorema cinese del resto), noi invece useremo semplice metodo a tentativi:
  Esempio:

  $d = 2$ -> $2.3mod40 = 6$ NO
  $d = 3$ -> $3.3mod40 = 9$ NO
  $d = 4$ -> $4.3mod40 = 12$ NO
  ...
  $d = 26$ -> $26.3mod40 = 78mod40 = 38$ NO
  $d = 27$ -> $27.3mod40 = 81mod40 = 1$ SI
  $d = 27$

Certificati e firma digitale

- La firma digitale è un sistema di cifratura basata sulla crittografia asimmetrica di documenti elettronici che garantisce l'autenticità dell'autore e la validità del contenuto, è basata su un certificato elettronico, che è un documento digitale che identifica l'autore della firma.
- Il certificato digitale è rilasciato da un'autorità di certificazione (CA), che è un'organizzazione che verifica l'identità dell'utente e garantisce la validità del certificato ed è responsabile della gestione e del controllo dei certificati digitali.

I certificati digitali sono utilizzati per garantire la sicurezza delle transazioni elettroniche, come la firma di contratti, la trasmissione di dati sensibili e la gestione di documenti legali. La firma digitale è considerata legale e vincolante in molti paesi, compresi gli Stati Uniti e l'Unione Europea.

La firma digitale è un modo sicuro e affidabile per garantire l'autenticità e la validità dei documenti elettronici. Grazie ai certificati digitali, gli utenti possono essere certi che un documento è stato firmato da una persona specifica e che il contenuto del documento non è stato alterato in alcun modo.

RSA è uno degli algoritmi di crittografia a chiave pubblica più utilizzati per la firma digitale a causa della sua sicurezza e della sua efficienza. Tuttavia, ci sono anche altri algoritmi di crittografia a chiave pubblica che possono essere utilizzati per implementare la firma digitale, come ad esempio il DSA (Digital Signature Algorithm) e l'ECDSA (Elliptic Curve Digital Signature Algorithm).

La firma digitale viene solitamente salvata in un file separato con estensione ".p7s" o ".p7m", a seconda del formato utilizzato. Il formato ".p7s" è utilizzato per la firma digitale con crittografia asimmetrica (ad esempio, RSA), mentre il formato ".p7m" è utilizzato per la firma digitale con crittografia simmetrica.

HTTP Request e HTTP Response

L'HTTP (Hypertext Transfer Protocol) è un protocollo di comunicazione che permette il trasferimento di dati tra il client (ad esempio il browser web) e il server web.

- L'HTTP Request è il messaggio inviato dal client al server, contenente informazioni sulla risorsa richiesta, il metodo di richiesta (GET, POST, PUT, DELETE, ecc.) e altri parametri come i dati del form.
- L'HTTP Response è invece la risposta del server alla richiesta inviata dal client. Essa

contiene i dati richiesti, come ad esempio il contenuto di una pagina web, e informazioni supplementari come il codice di stato HTTP, che indica lo stato del processo della richiesta.

Esistono molti codici di stato HTTP, tra cui il più comune è il codice 200 OK, che indica che la richiesta è stata completata con successo. Altri codici di stato comuni includono il 404 Not Found, che indica che la risorsa richiesta non è stata trovata, o il 500 Internal Server Error, che indica un errore interno del server.

Sintassi di una richiesta HTTP:

```
VERBO Risorsa HTTP/Versione
Intestazioni
Corpo del messaggio
```

Esempio di **richiesta HTTP GET**:

```
GET /pagina.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/78.0.3904.97 Safari/537.36
Accept: text/html,application/xhtml+xml
```

Sintassi di una risposta HTTP:

```
Versione Codice di stato Frase di stato
Intestazioni
Corpo del messaggio
```

Esempio di **risposta HTTP**:

```
HTTP/1.1 200 OK
Date: Tue, 6 Jul 2023 12:00:00 GMT
Server: Apache/2.4.7 (Ubuntu)
Content-Length: 1234
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
```

```
    <title>Pagina di esempio</title>
</head>
<body>
    <h1>Benvenuto!</h1>
    <p>Questa è una pagina di esempio.</p>
</body>
</html>
```

Nella sintassi di una richiesta HTTP, il "VERBO" indica l'azione da compiere sulle risorse specificate. Alcuni esempi comuni di verbi sono GET, POST, PUT, DELETE, ecc. La "Risorsa" specifica il percorso o l'URL della risorsa richiesta. La "Versione" indica la versione del protocollo HTTP utilizzata.

Nella sintassi di una risposta HTTP, la "Versione" specifica la versione del protocollo utilizzata. Il "Codice di stato" indica lo stato della richiesta, ad esempio 200 per una risposta di successo, 404 per una risorsa non trovata, ecc. La "Frase di stato" fornisce una breve descrizione del codice di stato.

Le "Intestazioni" contengono informazioni aggiuntive sul messaggio, come i parametri di autenticazione, i cookie, il tipo di contenuto, ecc. Il "Corpo del messaggio" contiene il contenuto effettivo della richiesta o della risposta.

Header, Metodi e Codici di Stato HTTP

HTTP, acronimo di Hypertext Transfer Protocol, è un protocollo di comunicazione utilizzato per lo scambio di dati tra il client e il server su una rete di computer. In questo documento, verranno trattati i tre aspetti chiave di HTTP: header, metodi e codici di stato.

Header

L'header HTTP è una sezione dell'intera richiesta o della risposta, che contiene informazioni aggiuntive sulle richieste e sulle risposte. Ci sono diversi tipi di header HTTP, come l'header generale, l'header della richiesta, l'header della risposta e l'header dell'entità. L'header generale viene utilizzato per le informazioni che si applicano all'intera richiesta o risposta. L'header della richiesta viene utilizzato per le informazioni specifiche della richiesta inviata al server. L'header della risposta viene utilizzato per le informazioni specifiche della risposta ricevuta dal server. L'header dell'entità viene utilizzato per le informazioni relative all'entità inviata o ricevuta.

Metodi

I metodi HTTP sono i comandi utilizzati per indicare l'azione richiesta dal client. I metodi HTTP più comuni sono GET, POST, PUT, DELETE, HEAD e OPTIONS. Il metodo GET viene utilizzato per richiedere una risorsa specifica dal server. Il metodo POST viene utilizzato per inviare dati al server per essere elaborati. Il metodo PUT viene utilizzato per aggiornare una risorsa specifica sul server. Il metodo DELETE viene utilizzato per eliminare una risorsa specifica dal server. Il metodo HEAD viene utilizzato per richiedere le informazioni sull'header di una risorsa specifica senza richiedere il corpo della risposta. Il metodo OPTIONS viene utilizzato per richiedere le opzioni di comunicazione disponibili per una risorsa specifica.

Codici di Stato

I codici di stato HTTP vengono utilizzati per indicare lo stato della richiesta o della risposta. Ci sono diversi codici di stato HTTP, come 1xx, 2xx, 3xx, 4xx e 5xx. I codici di stato 1xx vengono utilizzati per indicare una risposta informativa. I codici di stato 2xx vengono utilizzati per indicare una risposta di successo. I codici di stato 3xx vengono utilizzati per indicare una reindirizzamento della richiesta. I codici di stato 4xx vengono utilizzati per indicare un errore nella richiesta inviata dal client. I codici di stato 5xx vengono utilizzati per indicare un errore generato dal server durante l'elaborazione della richiesta.

Il modello client-server

Il modello client-server è un'architettura di rete che suddivide le funzionalità dei programmi in due parti distinte: il client e il server.

- Il client è il programma o l'applicazione che viene eseguito sul computer dell'utente finale.
- Il server è il programma o l'applicazione che viene eseguito su un computer dedicato, noto come server, che fornisce un servizio o una risorsa al client.

Il modello client-server è stato progettato per migliorare l'efficienza delle comunicazioni in rete e per garantire una maggiore sicurezza dei dati. In questo modello, il client non ha accesso diretto alle risorse del server, ma deve fare richieste attraverso una connessione di rete. Il server, a sua volta, risponde alle richieste del client e fornisce solo

le informazioni richieste.

Il modello client-server è utilizzato in una vasta gamma di applicazioni, come ad esempio nei siti web, nei servizi di posta elettronica, nei giochi online e nei sistemi di gestione di database. In questo modo, i client possono accedere alle informazioni e alle risorse del server senza dover disporre di hardware o software specifici.

Inoltre, il modello client-server consente di gestire meglio le risorse della rete, poiché le richieste dei client vengono elaborate dal server in modo centralizzato e gestito da un amministratore di sistema. In questo modo, è possibile garantire un maggiore controllo e una maggiore sicurezza delle informazioni scambiate sulla rete.

Funzionamento

1. il client manda una richiesta al server.
2. il server (in attesa) riceve la richiesta.
3. il server esegue il servizio richiesto (generando un thread concorrente).
4. il server manda una risposta ed eventualmente dei dati.
5. il client riceve la risposta ed eventualmente i dati.

Esempi di servizi client - server

- Telnet: mediante un tale programma (programma client) è possibile operare su un computer remoto come si opera su un computer locale; questo è possibile se sulla macchina remota è presente un programma server in grado di esaudire le richieste del client telnet.
- HTTP: il browser è un client http (Web), che richiede pagine Web ai computer su cui è installato un Web server, il quale esaudirà le richieste spedendo la pagina desiderata.
- FTP: tramite un client FTP è possibile copiare e cancellare file su un computer remoto, purché qui sia presente un server FTP.
- Altri servizi di questo tipo sono SMTP, IMAP, NFS, NIS...

Caratteristiche delle applicazioni client e

delle applicazioni server

Le applicazioni client e le applicazioni server sono due tipi di software utilizzati per la gestione e l'elaborazione dei dati. Tuttavia, presentano alcune importanti differenze tra loro.

Applicazioni client

Le applicazioni client sono programmi che vengono eseguiti sul computer dell'utente finale e richiedono una connessione a un server per funzionare correttamente. Sono progettati per interagire con l'utente e fornire una interfaccia grafica per l'elaborazione dei dati. Alcune delle caratteristiche delle applicazioni client includono:

- **Interfaccia utente** - Le applicazioni client sono progettate per fornire all'utente una interfaccia grafica per l'elaborazione dei dati. Ciò significa che l'utente può interagire con l'applicazione utilizzando la tastiera, il mouse o altri dispositivi di input.
- **Elaborazione locale** - Le applicazioni client utilizzano le risorse del computer dell'utente per l'elaborazione dei dati. Ciò significa che il computer dell'utente deve essere dotato di una quantità sufficiente di memoria e di una CPU abbastanza potente per eseguire l'applicazione correttamente.
- **Connessione remota** - Le applicazioni client richiedono una connessione a un server per funzionare correttamente. Ciò significa che l'utente deve avere una connessione Internet stabile e veloce per utilizzare l'applicazione.
- **Installazione locale** - Le applicazioni client devono essere installate sul computer dell'utente prima di poter essere utilizzate. Ciò significa che l'utente deve avere i permessi di amministratore per installare l'applicazione.

Applicazioni server

Le applicazioni server sono programmi che vengono eseguiti su un server remoto e forniscono servizi ad altre applicazioni o utenti. Sono progettati per elaborare i dati in modo efficiente e fornire una solida infrastruttura per la gestione dei dati. Alcune delle caratteristiche delle applicazioni server includono:

- **Elaborazione remota** - Le applicazioni server utilizzano le risorse del server per l'elaborazione dei dati. Ciò significa che il server deve essere dotato di una quantità

sufficiente di memoria e di una CPU abbastanza potente per eseguire l'applicazione correttamente.

- **Connessioni multiple** - Le applicazioni server possono gestire molte connessioni contemporaneamente. Ciò significa che possono fornire servizi a molti utenti contemporaneamente senza compromettere le prestazioni.
- **Installazione remota** - Le applicazioni server non richiedono l'installazione sul computer dell'utente. Ciò significa che l'utente può accedere all'applicazione tramite una connessione Internet senza dover installare alcun software.
- **Sicurezza** - Le applicazioni server dispongono di meccanismi di sicurezza per proteggere i dati degli utenti. Ciò significa che i dati degli utenti sono protetti da accessi non autorizzati.

Livelli e Strati (Applicazioni Two-Tier, Three-Tier, Multi-Tier)

Nella progettazione di applicazioni software, un aspetto fondamentale è la definizione dei livelli e degli strati che compongono il sistema. In particolare, la suddivisione in livelli e strati consente di separare le diverse funzionalità dell'applicazione, semplificando la gestione e la manutenzione del codice.

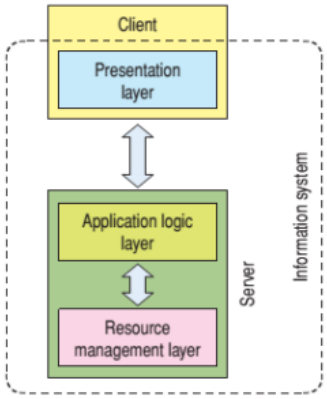
Architettura One-Tier

- A partire dagli anni Settanta, le architetture si riducevano a un solo mainframe al quale erano collegati i terminali “stupidi”.
- Tutta l’elaborazione era effettuata dall’elaboratore centrale e i terminali servivano solo per le fasi di I/O.
- Questa architettura non rientra nella tipologia client-server e può essere classificata come architettura a un solo livello (1 tier) ed è la situazione che si presentava prima dell’avvento dei sistemi distribuiti.

Applicazioni Two-Tier

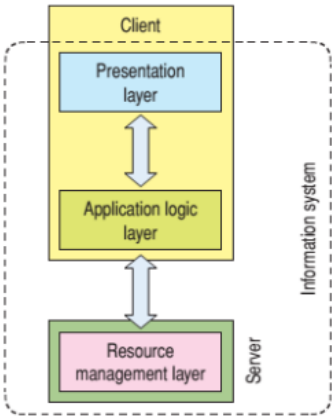
Le applicazioni Two-Tier, o a due livelli, sono caratterizzate dalla presenza di due strati principali: il client e il server.

- Il modello thin-client:
  - Il server è responsabile della logica applicativa e gestione dei dati
  - Il client è responsabile della esecuzione del software di presentazione.



Architettura a due livelli – 2 tier – modello thin

- Il modello thick-client (o fat-client):
  - Il server è responsabile della gestione dei dati.
  - Il client è responsabile di presentazione e logica applicativa.



Architettura a due livelli – 2 tier – modello thick

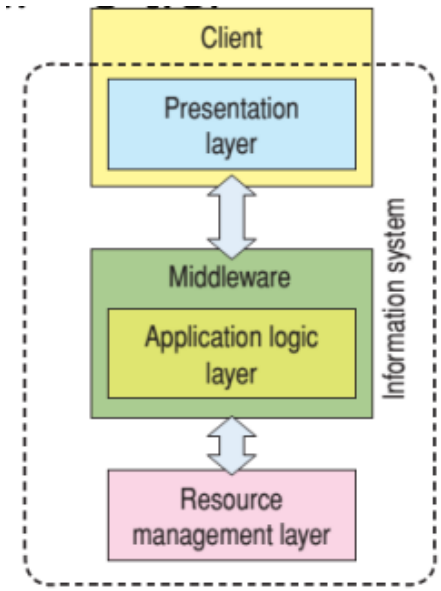
Mentre il modello thin-client è stato il primo passo per effettuare il passaggio dai sistemi mainframe alle architetture distribuite, con il thick-client si è spostata parte della applicazione sul client favorendo la connessione di host di tipo diverso che era di fatto praticamente impossibile nei modelli precedenti.

In questo tipo di architettura, quindi, la gestione dei dati è centralizzata e affidata al server.

Applicazioni Three-Tier

Le applicazioni Three-Tier, o a tre livelli, prevedono la presenza di un ulteriore strato intermedio rispetto alle applicazioni Two-Tier. Questo strato, chiamato server di applicazione o application server, ha il compito di gestire la logica di business e l'accesso ai dati. Il client, invece, si occupa solo dell'interfaccia utente e della presentazione dei dati, mentre il database è affidato al terzo strato.

- front-end o presentation tier: è l'interfaccia verso l'utente.
- logica applicativa o middle tier (business-tier).
- back-end con l'accesso alle risorse/ai dati, anche detto data tier (o resource-tier).



Architettura a tre livelli – 3 tier

Applicazioni Multi-Tier

Le applicazioni Multi-Tier, o a più livelli, prevedono la presenza di più strati intermedie rispetto alle applicazioni Three-Tier. Questa architettura è utilizzata per gestire applicazioni di grandi dimensioni e complessità, in cui è necessario suddividere le funzionalità in diverse componenti. In questo caso, è possibile suddividere le funzionalità in moduli separati, ciascuno dei quali può essere gestito da un server di applicazione dedicato.

HTML e CSS - Javascript

- HTML e CSS sono i linguaggi di markup e di stile utilizzati per creare pagine web. Javascript, invece, è un linguaggio di programmazione utilizzato per creare interazioni dinamiche all'interno di una pagina web.
- Con HTML, possiamo creare la struttura di una pagina web, come il titolo, i paragrafi, le immagini e i collegamenti. CSS, invece, ci permette di definire lo stile della pagina, come il colore del testo, il tipo di carattere e la posizione degli elementi.
- Javascript, invece, ci permette di creare interazioni dinamiche all'interno di una

pagina web, come ad esempio la verifica dei campi di un modulo prima dell'invio dei dati, la creazione di effetti visivi, l'aggiornamento di parti della pagina senza doverla ricaricare completamente e molto altro ancora.

- Insieme, HTML, CSS e Javascript permettono di creare pagine web moderne e interattive, in grado di offrire un'esperienza utente coinvolgente e di alta qualità.

Esempio HTML / CSS

```
<!DOCTYPE html>
<html>
  <head>
    <title>La mia pagina web</title>
    <link rel="stylesheet" type="text/css" href="stile.css">
  </head>
  <body>
    <header>
      <h1>Benvenuti nella mia pagina web</h1>
      <nav>
        <ul>
          <li><a href="#">Home</a></li>
          <li><a href="#">Chi siamo</a></li>
          <li><a href="#">Contatti</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <section>
        <h2>La nostra storia</h2>
        <p>Esempio storia.</p>
      </section>
      <section>
        <h2>Il nostro team</h2>
        <p>Il nostro team è composto da professionisti altamente qualificati.</p>
      </section>
    </main>
    <footer>
      <p>Copyright &copy; 2023. Tutti i diritti riservati.</p>
    </footer>
  </body>
</html>
```

```
/* Stile per il body */
body {
  font-family: Arial, sans-serif;
```

```
  margin: 0;
  padding: 0;
}

/* Stile per l'header */
header {
  background-color: #333;
  color: #fff;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 20px;
}

/* Stile per il logo */
header h1 {
  margin: 0;
}

/* Stile per la navigazione */
nav ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  display: flex;
}

nav li {
  margin: 0 10px;
}

nav a {
  color: #fff;
  text-decoration: none;
}

/* Stile per le sezioni */
section {
  padding: 20px;
}

/* Stile per il footer */
footer {
  background-color: #333;
  color: #fff;
  padding: 10px 20px;
  text-align: center;
}
```

In questo esempio, il codice HTML definisce una pagina web con un header, un main e

un footer. Il tag `<link>` nella sezione head viene utilizzato per importare il file CSS "stile.css", che definisce lo stile visivo della pagina.

Il codice CSS definisce lo stile visivo della pagina, come i colori di sfondo, il font e il layout della pagina. In questo esempio, il CSS viene utilizzato per definire lo stile dell'header, della navigazione, delle sezioni e del footer.

Questo è solo un esempio molto semplice, ma HTML e CSS possono essere utilizzati per creare pagine web molto più complesse e interattive.

META TAG

Le **meta tag in HTML** sono utilizzate per fornire informazioni aggiuntive sul documento HTML ai browser web e ad altri programmi che analizzano la pagina. Le meta tag possono fornire informazioni sul titolo del documento, la descrizione, le parole chiave, l'autore, la lingua, il carattere utilizzato, la visualizzazione dei dispositivi mobili e molto altro. Di seguito vediamo come utilizzare alcune meta tag per adattare il sito web a diversi dispositivi.

1. Viewport
   La meta tag `viewport` consente di impostare le proprietà di visualizzazione del contenuto in base alle dimensioni dello schermo del dispositivo. Ad esempio, la seguente meta tag imposta la larghezza del viewport sul dispositivo in modo che corrisponda alla larghezza del dispositivo:

   ```
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   ```

2. Charset
   La meta tag charset definisce il set di caratteri utilizzato nella pagina HTML. Ad esempio, la seguente meta tag definisce l'utilizzo del set di caratteri UTF-8:

   ```
   <meta charset="UTF-8">
   ```

3. Descrizione e keywords
   Le meta tag `description` e `keywords` forniscono informazioni sul contenuto della pagina. La meta tag `description` fornisce una breve descrizione della pagina, mentre la meta tag `keywords` elenca le parole chiave che descrivono la pagina. Ad esempio:

```
<meta name="description" content="Un sito web che offre tutorial e guide per imparare a progr
ammare">
<meta name="keywords" content="programmazione, coding, tutorial, guide">
```

4. Icona del sito
   La meta tag `rel="icon"` definisce l'icona del sito web. Ad esempio, la seguente meta tag imposta l'icona del sito web come "favicon.ico":

```
<meta rel="icon" href="favicon.ico" type="image/x-icon">
```

5. CSS per dispositivi mobili
   La meta tag link con il media `attribute="only screen and (max-width: 600px)"` consente di applicare specifici stili CSS ai dispositivi mobili con una larghezza massima di 600px. Ad esempio:

```
<link rel="stylesheet" href="mobile.css" media="only screen and (max-width: 600px)">
```

MEDIA QUERY

Le **media query in CSS** consentono di definire regole di stile specifiche per determinati dispositivi o dimensioni dello schermo. In questo modo è possibile adattare il sito web a diversi dispositivi, come computer desktop, tablet o smartphone. Le media query in CSS sono utilizzate insieme alle meta tag in HTML per fornire un'esperienza di navigazione ottimale su tutti i dispositivi.

Le media query in CSS sono basate sulla sintassi `@media` . La sintassi di base è la seguente:

```
@media media-type and (media-feature-rule) {
  /* Regole di stile per dispositivi specifici */
}
```

La `media type` specifica il tipo di dispositivo, come " `all` " (tutti i dispositivi), " `screen` " (schermi di computer), " `print` " (stampa) e così via. Il media `feature` rule specifica la regola specifica, come la larghezza dello schermo o l'orientamento.

Di seguito vediamo alcuni esempi di media query in CSS per adattare il sito a diversi

dispositivi:

1. Adattare il sito web ai dispositivi mobili con larghezza inferiore a 600px:

```
@media only screen and (max-width: 600px) {
  /* Regole di stile per dispositivi mobili */
}
```

2. Adattare il sito web ai dispositivi con una larghezza di schermo compresa tra 600px e 1024px:

```
@media only screen and (min-width: 600px) and (max-width: 1024px) {
  /* Regole di stile per tablet */
}
```

3. Adattare il sito web ai dispositivi con una larghezza di schermo superiore a 1024px:

```
@media only screen and (min-width: 1024px) {
  /* Regole di stile per computer desktop */
}
```

4. Adattare il sito web alla modalità notturna:

```
@media (prefers-color-scheme: dark) {
  /* Regole di stile per la modalità notturna */
}
```

In questo esempio, la meta tag in HTML `<meta name="color-scheme" content="dark light">` definisce la modalità di colori supportata dal sito web.

Esempio JavaScript

```
function greet(name) {
  console.log("Hello, " + name + "!");
}

greet("John");
```

Questo codice definisce una funzione `greet` che prende un parametro `name` e stampa una stringa di saluto che include il nome passato come parametro. La funzione viene poi chiamata con il nome "John".

Ora analizziamo la sintassi del codice riga per riga:

- `function greet(name) {` : questa riga definisce una funzione chiamata `greet` che ha un parametro chiamato `name` . Il corpo della funzione è racchiuso tra le parentesi graffe `{}` .
- `console.log("Hello, " + name + "!");` : questa riga stampa una stringa che include il testo "Hello, ", il valore del parametro `name` e il carattere "!" utilizzando l'operatore di concatenazione `+` . La funzione `console.log` viene utilizzata per stampare il risultato sulla console del browser.
- `greet("John");` : questa riga chiama la funzione `greet` passando il valore "John" come argomento per il parametro `name` .

In generale, la sintassi di Javascript è simile ad altri linguaggi di programmazione come Java e C++. Le funzioni vengono definite con la parola chiave `function` seguita dal nome della funzione e dai parametri racchiusi tra parentesi tonde `()` . Il corpo della funzione è racchiuso tra parentesi graffe `{}` . Le variabili vengono dichiarate utilizzando le parole chiave `var` , `let` o `const` . Gli operatori aritmetici e logici sono simili ad altri linguaggi e includono `+` , `-` , `*` , `/` , `%` , `&&` , `||` e `!` .

Inoltre, Javascript è un linguaggio interpretato, quindi non è necessario compilare il codice prima di eseguirlo. Il codice viene eseguito direttamente dal browser o dal motore Javascript sul server.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Esempio JavaScript in HTML</title>
  </head>
  <body>
    <h1>Calcolatrice</h1>
    <input type="number" id="num1" placeholder="Inserisci un numero">
    <input type="number" id="num2" placeholder="Inserisci un altro numero">
    <br><br>
    <button onclick="calcola()">Calcola</button>
    <br><br>
    <div id="risultato"></div>
```

```
    <script>
      function calcola() {
        const num1 = document.getElementById("num1").value;
        const num2 = document.getElementById("num2").value;
        const risultato = parseInt(num1) + parseInt(num2);
        document.getElementById("risultato").innerHTML = "Il risultato è: " + risultato;
      }
    </script>
  </body>
</html>
```

In questo esempio, abbiamo creato una semplice pagina HTML che contiene una calcolatrice. Abbiamo incluso il codice JavaScript all'interno del tag `<script>` all'interno del documento HTML. La funzione `calcola()` definita in JavaScript prende i valori inseriti negli elementi di input, li somma e visualizza il risultato nell'elemento con id "risultato". Abbiamo anche associato la funzione `calcola()` a un pulsante tramite l'attributo `onclick`.

Nota che questo è solo un esempio di come potresti utilizzare il codice JavaScript generato in una pagina HTML. In generale, il codice JavaScript viene incluso in un file separato con estensione `.js` e viene collegato alla pagina HTML tramite il tag `<script>` con l'attributo `src`.

getElementById()

La funzione `getElementById()` è un metodo disponibile nell'oggetto `document` in JavaScript, e viene utilizzata per ottenere un riferimento a un elemento HTML all'interno di un documento tramite il suo attributo "id".

Ecco un esempio di come utilizzare `getElementById()` per ottenere un riferimento a un elemento e manipolarlo:

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function changeText() {
      // Ottenere un riferimento all'elemento utilizzando getElementById()
      var element = document.getElementById("myElement");

      // Modificare il contenuto dell'elemento
      element.innerHTML = "Nuovo testo!";
    }
  </script>
```

```
</head>
<body>

  <h2>Esempio di utilizzo di getElementById()</h2>

  <p id="myElement">Testo originale</p>

  <button onclick="changeText()">Cambia Testo</button>

</body>
</html>
```

Nell'esempio sopra, abbiamo un paragrafo con l'id "myElement" che contiene del testo di esempio. Quando viene cliccato il pulsante "Cambia Testo", la funzione `changeText()` viene chiamata.

All'interno della funzione, utilizziamo `document.getElementById("myElement")` per ottenere un riferimento all'elemento con l'id "myElement" all'interno del documento HTML. Assegniamo il riferimento a una variabile chiamata `element`.

Successivamente, utilizziamo `element.innerHTML` per accedere e modificare il contenuto HTML dell'elemento. In questo caso, impostiamo il nuovo testo "Nuovo testo!" come contenuto dell'elemento.

Quando esegui questo esempio nel tuo browser e fai clic sul pulsante, vedrai il testo all'interno del paragrafo cambiare in "Nuovo testo!".

L'utilizzo di `getElementById()` è molto comune quando si desidera ottenere un riferimento a un elemento specifico all'interno di un documento HTML e manipolarne il contenuto o gli attributi tramite JavaScript.

.innerHTML

La proprietà `innerHTML` in JavaScript viene utilizzata per ottenere o impostare il contenuto HTML di un elemento. Essa permette di manipolare il markup HTML all'interno di un elemento specifico della pagina.

Ecco un esempio che illustra come utilizzare la proprietà `innerHTML` per ottenere e impostare il contenuto di un elemento:

```
<!DOCTYPE html>
<html>
<head>
```

```
  <script>
    function changeContent() {
      // Ottenere il contenuto HTML di un elemento
      var paragraph = document.getElementById("myParagraph");
      var content = paragraph.innerHTML;
      console.log("Contenuto attuale: " + content);

      // Impostare il contenuto HTML di un elemento
      var newContent = "<strong>Nuovo contenuto</strong>";
      paragraph.innerHTML = newContent;
      console.log("Nuovo contenuto impostato.");
    }
  </script>
</head>
<body>

  <h2>Esempio di utilizzo di .innerHTML</h2>

  <p id="myParagraph">Contenuto originale</p>

  <button onclick="changeContent()">Cambia Contenuto</button>

</body>
</html>
```

Nell'esempio sopra, abbiamo un paragrafo con l'id "myParagraph" che contiene del testo di esempio. Quando viene cliccato il pulsante "Cambia Contenuto", la funzione `changeContent()` viene chiamata.

All'interno della funzione, utilizziamo `document.getElementById("myParagraph")` per ottenere il riferimento all'elemento paragrafo. Successivamente, utilizziamo `paragraph.innerHTML` per ottenere il contenuto HTML corrente dell'elemento e lo visualizziamo nella console utilizzando `console.log()`.

Successivamente, creiamo una nuova stringa HTML `<strong>Nuovo contenuto</strong>` e la assegniamo alla proprietà `innerHTML` dell'elemento paragrafo. In questo modo, il contenuto dell'elemento viene sostituito con il nuovo contenuto.

Infine, visualizziamo un messaggio nella console per indicare che il nuovo contenuto è stato impostato.

Quando esegui questo esempio nel tuo browser e fai clic sul pulsante, vedrai il contenuto del paragrafo cambiare e i messaggi di log verranno visualizzati nella console del browser.

RegEX

Una regex, abbreviazione di "espressione regolare" o "regular expression" in inglese, è una sequenza di caratteri che definisce un pattern di ricerca all'interno di un testo. Le regex sono ampiamente utilizzate per la ricerca, l'elaborazione e la manipolazione di stringhe di testo in diversi linguaggi di programmazione e strumenti.

Una regex può essere composta da caratteri normali, che corrispondono esattamente a se stessi, e da caratteri speciali che rappresentano classi di caratteri o metacaratteri. Ad esempio, il punto (.) in una regex corrisponde a qualsiasi carattere, mentre l'asterisco (*) rappresenta zero o più ripetizioni del carattere o del gruppo che lo precede.

Le regex permettono di specificare pattern complessi per la ricerca di corrispondenze in un testo. Possono essere utilizzate per controllare se una stringa rispetta un determinato formato, trovare parole o frasi specifiche, effettuare sostituzioni di testo e altro ancora. Sono molto potenti e flessibili, ma richiedono una certa conoscenza per essere utilizzate correttamente.

Ecco alcuni esempi di utilizzo delle regex:

- Trovare tutte le occorrenze di una parola in un testo.
- Verificare se un indirizzo email è valido.
- Sostituire tutte le occorrenze di una parola con un'altra.
- Estrarre parti specifiche di un testo, come numeri di telefono o indirizzi.

Le regex possono variare leggermente nella sintassi a seconda del linguaggio o dello strumento utilizzato, ma il concetto di base rimane lo stesso. È possibile utilizzare librerie o funzioni specifiche del linguaggio per eseguire operazioni con le regex, ad esempio in Python si può utilizzare il modulo "re" per lavorare con espressioni regolari.

Le regex sono uno strumento molto potente per la manipolazione di stringhe di testo e sono ampiamente utilizzate nella programmazione e nell'elaborazione dei dati.

Certamente! Ecco degli esempi di come utilizzare le espressioni regolari in JavaScript, PHP e C#:

JavaScript:

```
var regex = /pattern/; // Creazione di un'istanza di RegExp
var string = "Testo di esempio";
```

```
var result = regex.test(string); // Verifica della corrispondenza nel testo
console.log(result); // Output: true o false

var match = string.match(regex); // Trova la corrispondenza nel testo
console.log(match); // Output: array contenente le corrispondenze trovate
```

PHP:

```
$regex = "/pattern/"; // Creazione di un'istanza di regex
$string = "Testo di esempio";
$result = preg_match($regex, $string); // Verifica della corrispondenza nel testo
var_dump($result); // Output: 1 o 0

$matches = [];
preg_match_all($regex, $string, $matches); // Trova tutte le corrispondenze nel testo
print_r($matches); // Output: array contenente le corrispondenze trovate
```

C#:

```
using System;
using System.Text.RegularExpressions;

string regexPattern = @"pattern"; // Creazione di un'istanza di regex
string input = "Testo di esempio";
bool isMatch = Regex.IsMatch(input, regexPattern); // Verifica della corrispondenza nel testo
Console.WriteLine(isMatch); // Output: True o False

Match match = Regex.Match(input, regexPattern); // Trova la corrispondenza nel testo
if (match.Success)
{
    Console.WriteLine(match.Value); // Output: la corrispondenza trovata
}
```

In questi esempi, "pattern" rappresenta l'espressione regolare che desideri utilizzare per cercare corrispondenze nel testo. Le variabili "string" o "input" contengono il testo in cui effettuare la ricerca.

Nella maggior parte dei casi, le funzioni o i metodi specifici per le espressioni regolari richiedono un'istanza di regex e un input su cui applicare la ricerca. Puoi utilizzare diverse funzioni o metodi a seconda dell'uso desiderato, come `test()` o `match()` in JavaScript, `preg_match()` o `preg_match_all()` in PHP e `Regex.IsMatch()` o `Regex.Match()` in C#.

Protocolli SOAP, REST

I protocolli SOAP e REST sono entrambi utilizzati per la comunicazione tra applicazioni web, ma differiscono nella loro architettura e funzionalità.

- SOAP (Simple Object Access Protocol) è un protocollo basato su XML che utilizza una struttura complessa per la comunicazione tra applicazioni web. SOAP richiede l'utilizzo di un WSDL (Web Services Description Language) per definire la struttura dei dati scambiati tra le applicazioni e utilizza il protocollo HTTP per la trasmissione dei dati.
- REST (Representational State Transfer) è invece un protocollo più semplice basato sull'utilizzo delle richieste HTTP GET, POST, PUT e DELETE. REST utilizza una comunicazione stateless, ovvero ogni richiesta contiene tutte le informazioni necessarie per l'esecuzione dell'operazione richiesta. REST non richiede un WSDL per la definizione della struttura dei dati.

In sintesi, SOAP è più complesso ma offre una maggiore flessibilità nella definizione della struttura dei dati, mentre REST è più semplice e veloce ma meno flessibile nella definizione della struttura dei dati.

- Ecco un esempio di utilizzo di SOAP:

  Supponiamo di voler sviluppare un'applicazione che consenta agli utenti di ottenere informazioni sui prezzi dei prodotti di un negozio online. In questo caso, potremmo utilizzare il protocollo SOAP per accedere ai servizi web dell'applicazione del negozio online. Ad esempio, potremmo creare una richiesta SOAP che includa il nome del prodotto e il codice del paese, e inviarla al servizio web del negozio online per ottenere il prezzo del prodotto in quel paese.

- Ecco un esempio di utilizzo di REST:

  Supponiamo di voler sviluppare un'applicazione che consenta agli utenti di cercare le recensioni degli utenti sui ristoranti di una città. In questo caso, potremmo utilizzare il protocollo REST per accedere ai dati del servizio di recensioni online. Ad esempio, potremmo creare una richiesta HTTP GET che includa l'indirizzo del ristorante e inviarla al servizio web del servizio di recensioni per ottenere le recensioni degli utenti per quel ristorante.

In sintesi, SOAP è un protocollo basato su XML e utilizzato principalmente per la

comunicazione tra applicazioni aziendali, mentre REST è un protocollo basato su HTTP e utilizzato principalmente per la comunicazione tra applicazioni web. La scelta tra i due dipende dalle specifiche esigenze del progetto e dal tipo di dati e di applicazioni coinvolte.

XML e JSON; AJAX

XML e JSON sono due formati di dati utilizzati per lo scambio di informazioni tra diversi sistemi. Entrambi i formati sono basati su testo e sono ampiamente utilizzati per la trasmissione di dati su Internet.

- XML (Extensible Markup Language) è un formato di dati strutturato che consente di definire tag personalizzati e strutture di dati complesse. XML è stato sviluppato per la condivisione di dati strutturati sui sistemi di rete e viene utilizzato per la creazione di documenti HTML, RSS e molti altri formati di dati.
- JSON (JavaScript Object Notation) è un formato di dati basato su JavaScript e viene utilizzato per la trasmissione di dati tra client e server. JSON è facile da leggere e scrivere e viene utilizzato in molte applicazioni Web per lo scambio di dati.
- AJAX (Asynchronous JavaScript and XML) è una tecnologia Web che consente di caricare dati in modo asincrono senza dover aggiornare l'intera pagina. AJAX utilizza JavaScript e XML o JSON per effettuare richieste al server e aggiornare solo le parti della pagina che devono essere modificate.

In sintesi, XML e JSON sono formati di dati utilizzati per lo scambio di informazioni tra i sistemi e AJAX è una tecnologia Web che utilizza questi formati per caricare dati in modo asincrono.

JSON

JSON è un formato di scambio dati leggero e facile da leggere e scrivere per le macchine e gli umani. È basato sulla sintassi degli oggetti JavaScript, ma è un formato di testo indipendente dal linguaggio di programmazione. Questo lo rende un formato molto popolare per lo scambio di dati tra applicazioni web.

JSON è costituito da coppie chiave-valore, dove ogni campo viene separato da una virgola e racchiuso tra parentesi graffe. La sintassi è molto simile a quella degli oggetti JavaScript. Qui di seguito un esempio di un oggetto JSON:

```
{
   "name": "Mario Rossi",
   "age": 30,
   "city": "Roma"
}
```

È possibile utilizzare JSON per rappresentare dati strutturati come array, oggetti, numeri, stringhe, booleani e null. Questo lo rende un formato molto flessibile e adatto a molte applicazioni.

JSON è anche facile da usare con molti linguaggi di programmazione. La maggior parte dei linguaggi di programmazione moderni ha librerie per la lettura e la scrittura di dati JSON. Questo lo rende un formato molto popolare per lo scambio di dati tra diverse applicazioni.

In definitiva, JSON è un formato di scambio dati molto popolare e utile per lo sviluppo di applicazioni web. Grazie alla sua leggibilità e flessibilità, è stato adottato da molte aziende e organizzazioni come formato di scambio dati predefinito.

Esempio JSON

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script lang="javascript">
            document.write("<h1>Esempio JSON</h1>");
            document.write("<br>")
        var oggetto1 =
        {
            "nome" : "Libro 1",
            "linguaggio" : "Java" ,
            "autore" : "JKR" ,
        };
        document.write("<h2>Nome:"+ oggetto1.nome +" </h2>");
        document.write("<h2>Linguaggio: "+ oggetto1.linguaggio +" </h2>");
        document.write("<h2>Autore: "+ oggetto1.autore +" </h2>");
        document.write("<br>");
        var oggetto2 =
        {
            "nome" : "Libro 2",
            "linguaggio" : "C#" ,
            "autore" : "Maurizio Merluzzo" ,
        };
```

```
        document.write("<h2>Nome:"+ oggetto2.nome +" </h2>");
        document.write("<h2>Linguaggio: "+ oggetto2.linguaggio +" </h2>");
        document.write("<h2>Autore: "+ oggetto2.autore +" </h2>");
        document.write("<br>")
    </script>
</head>
<body>

</body>
</html>
```

Questo è un codice HTML che contiene anche del codice JavaScript per creare oggetti JSON e stamparli a schermo.

<!DOCTYPE html> indica la versione del documento HTML utilizzata, mentre <html lang="en"> indica la lingua del documento.
Nel tag <head> sono presenti alcune informazioni sul documento, come la codifica dei caratteri (<meta charset="UTF-8">) e del codice JavaScript.

La sezione di codice JavaScript inizia con <script lang="javascript"> e termina con </script>. In questo esempio viene creato un oggetto JSON (oggetto1) che contiene le proprietà "nome", "linguaggio" e "autore". Queste proprietà vengono stampate a schermo utilizzando la funzione document.write().

Viene poi creato un secondo oggetto JSON (oggetto2) con proprietà simili, e anche queste vengono stampate a schermo utilizzando la funzione document.write().

Il risultato finale sarà una pagina HTML con due oggetti JSON visualizzati a schermo.

XML

XML (Extensible Markup Language) è un linguaggio di markup flessibile utilizzato per la creazione di documenti strutturati e scambi di dati tra diverse applicazioni. In pratica, XML fornisce una sintassi per definire e descrivere i dati in un formato leggibile da una vasta gamma di programmi e sistemi.

Gli elementi principali di XML sono le "tag", ovvero i tag di apertura e chiusura che definiscono gli elementi del documento. Ad esempio, un elemento XML potrebbe avere un tag di apertura `<nome>` e un tag di chiusura `</nome>`, all'interno dei quali si trova il contenuto dell'elemento, come il nome di una persona o un valore numerico.

Inoltre, XML consente di definire i propri tag personalizzati e di creare strutture dati complesse, come tabelle, grafici e documenti interattivi. XML è spesso utilizzato per lo

scambio di dati tra applicazioni di diversi fornitori, in quanto offre una sintassi standard e aperta che può essere facilmente interpretata da qualsiasi sistema.

Esistono molte tecnologie e linguaggi basati su XML, come ad esempio RSS, SOAP, XHTML e molti altri.

```
<?xml version="1.0" encoding="UTF-8"?>
<libro>
<titolo>Il signore degli anelli</titolo>
<autore>J.R.R. Tolkien</autore>
<genere>Fantasy</genere>
<anno>1954</anno>
<prezzo valuta="euro">15.99</prezzo>
</libro>
```

In questo esempio, il codice XML inizia con una dichiarazione che indica la versione del linguaggio e la codifica dei caratteri. Il documento principale è il tag `<libro>` che contiene diversi elementi, come il titolo del libro, l'autore, il genere, l'anno di pubblicazione e il prezzo. L'elemento `<prezzo>` contiene un attributo denominato "valuta" con il valore "euro".

L'uso degli elementi e degli attributi in XML è molto flessibile e può essere adattato alle esigenze specifiche di ciascun progetto.

AJAX

AJAX (Asynchronous JavaScript and XML) è una tecnologia di programmazione Web utilizzata per creare applicazioni web dinamiche e interattive senza dover ricaricare l'intera pagina web. In pratica, AJAX consente di inviare e ricevere dati in background da un server, senza interferire con la visualizzazione della pagina web.

Il principale vantaggio di AJAX è che consente agli utenti di interagire con una pagina web senza dover attendere il caricamento completo della pagina ogni volta che viene effettuata una richiesta. Ad esempio, quando si scrive un messaggio in una chat online, AJAX può inviare la richiesta al server in background e aggiornare la chat in tempo reale, senza dover ricaricare l'intera pagina web.

AJAX utilizza principalmente tre tecnologie: HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) e JavaScript. HTML e CSS sono utilizzati per la struttura e lo

stile della pagina web, mentre JavaScript è utilizzato per gestire le richieste e le risposte del server.

Il flusso di lavoro di una richiesta AJAX è il seguente: quando un utente interagisce con una pagina web, ad esempio facendo clic su un pulsante, il codice JavaScript invia una richiesta al server senza dover ricaricare la pagina. Il server elabora la richiesta e restituisce una risposta in formato XML, JSON o HTML. Infine, il codice JavaScript aggiorna la pagina web in base alla risposta del server, senza dover ricaricare l'intera pagina.

AJAX è utilizzato in molte applicazioni Web moderne, come le applicazioni di posta elettronica, le piattaforme di social media e molti altri siti web interattivi e dinamici.

```
// Creazione di un'istanza dell'oggetto XMLHttpRequest
var xhttp = new XMLHttpRequest();

// Definizione della funzione di gestione della risposta del server
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    // Aggiornamento del contenuto della pagina con la risposta del server
    document.getElementById("demo").innerHTML = this.responseText;
  }
};

// Invio della richiesta al server
xhttp.open("GET", "esempio.php", true);
xhttp.send();
```

In questo esempio, l'oggetto XMLHttpRequest viene utilizzato per inviare una richiesta GET al file "esempio.php" sul server. La funzione di gestione della risposta del server viene definita con l'attributo `onreadystatechange`, che si attiva ogni volta che lo stato dell'oggetto XMLHttpRequest cambia. Quando lo stato è 4 (richiesta completata) e lo status è 200 (risposta OK), la risposta del server viene visualizzata nell'elemento con l'id "demo" tramite la proprietà `innerHTML`

Utilizzo AJAX in html

1. Ecco un esempio di come puoi utilizzare Ajax in un documento HTML utilizzando

JavaScript:

```
<!DOCTYPE html>
<html>
<head>
  <script src="<https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js>"></script>
</head>
<body>

  <h2>Caricamento dinamico del contenuto con Ajax</h2>

  <button onclick="loadContent()">Carica Contenuto</button>

  <div id="content"></div>

  <script>
    function loadContent() {
      $.ajax({
        url: "pagina.php", // L'URL del tuo script PHP o servizio web
        method: "GET", // Metodo HTTP da utilizzare (GET, POST, etc.)
        success: function(response) {
          // Callback per gestire la risposta del server
          $("#content").html(response);
        },
        error: function(xhr, status, error) {
          // Callback in caso di errore
          console.log(error);
        }
      });
    }
  </script>

</body>
</html>
```

In questo esempio, abbiamo un pulsante "Carica Contenuto" e un div con l'id "content" che verrà utilizzato per visualizzare il contenuto ottenuto tramite Ajax.

Quando viene cliccato il pulsante, la funzione `loadContent()` viene chiamata. All'interno di questa funzione, viene utilizzata la funzione `$.ajax()` di jQuery per eseguire una richiesta Ajax. L'URL specificato nella proprietà `url` rappresenta l'indirizzo dell'endpoint del server a cui fare la richiesta. Può essere un file PHP, un servizio web o un'altra risorsa che elabora la richiesta e restituisce una risposta.

Nell'esempio, è stato specificato il metodo HTTP "GET" tramite la proprietà `method`, ma è possibile utilizzare anche altri metodi come "POST" o "PUT" a seconda delle esigenze.

La funzione `success` viene utilizzata come callback per gestire la risposta del server. In questo caso, viene utilizzato `$("#content").html(response)` per impostare il contenuto della risposta all'interno dell'elemento con id "content".

Se si verifica un errore durante la richiesta Ajax, la funzione `error` viene chiamata e viene visualizzato un messaggio di errore nella console del browser utilizzando `console.log(error)`.

È importante notare che nell'esempio viene utilizzata la libreria jQuery per semplificare l'utilizzo di Ajax, ma esistono anche altre librerie e metodi nativi in JavaScript per eseguire richieste Ajax.

2. Ecco un altro esempio di come utilizzare Ajax in un documento HTML senza l'uso di librerie esterne come jQuery:

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function loadContent() {
      var xhr = new XMLHttpRequest();
      xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
          var response = xhr.responseText;
          document.getElementById("content").innerHTML = response;
        } else if (xhr.readyState === 4 && xhr.status !== 200) {
          console.log("Si è verificato un errore durante la richiesta.");
        }
      };
      xhr.open("GET", "pagina.php", true);
      xhr.send();
    }
  </script>
</head>
<body>

  <h2>Caricamento dinamico del contenuto con Ajax</h2>

  <button onclick="loadContent()">Carica Contenuto</button>

  <div id="content"></div>

</body>
</html>
```

In questo esempio, abbiamo ancora un pulsante "Carica Contenuto" e un div con l'id "content" che verrà utilizzato per visualizzare il contenuto ottenuto tramite Ajax.

La funzione `loadContent()` viene chiamata quando viene cliccato il pulsante. All'interno di questa funzione, viene creato un oggetto XMLHttpRequest, che ci consente di effettuare richieste HTTP asincrone. Viene definito un gestore di eventi `onreadystatechange` per gestire i cambiamenti di stato della richiesta.

All'interno del gestore di eventi, verifichiamo se `xhr.readyState` è pari a 4 (che indica che la richiesta è stata completata) e `xhr.status` è uguale a 200 (che indica che la richiesta è stata completata con successo). Se entrambe le condizioni sono vere, otteniamo la risposta tramite `xhr.responseText` e impostiamo il contenuto nel div "content" utilizzando `document.getElementById("content").innerHTML = response`.

Se `xhr.readyState` è pari a 4 ma `xhr.status` non è 200, ciò indica che si è verificato un errore durante la richiesta. In questo caso, viene visualizzato un messaggio di errore nella console del browser utilizzando `console.log()`.

Infine, viene aperta la richiesta utilizzando `xhr.open()` specificando il metodo HTTP "GET" e l'URL dell'endpoint del server. La richiesta viene quindi inviata tramite `xhr.send()`.

NodeJs e ExpressJs

- **Node.js** è un ambiente di runtime JavaScript open-source basato sul motore JavaScript V8 di Google Chrome. Consente di eseguire JavaScript lato server, consentendo agli sviluppatori di creare applicazioni web e servizi di rete scalabili e ad alte prestazioni. A differenza del JavaScript che viene eseguito nel browser, Node.js può essere utilizzato per eseguire codice JavaScript sul server, consentendo di creare applicazioni web complete utilizzando solo JavaScript.
- **Express.js** è un framework web leggero e flessibile per Node.js. Fornisce una serie di funzionalità per la creazione di applicazioni web e API. Express.js semplifica il processo di gestione delle richieste HTTP, la gestione dei percorsi delle URL, la gestione dei middleware e delle risposte. È ampiamente utilizzato per creare backend per applicazioni web e servizi API.
    - Per **sviluppare API con Node.js e Express.js**, puoi seguire i seguenti passaggi:

1. Installa Node.js: Assicurati di avere Node.js installato sul tuo sistema. Puoi scaricarlo dal sito ufficiale di Node.js e seguire le istruzioni di installazione per il

tuo sistema operativo.

2. Crea un nuovo progetto: Crea una nuova cartella per il tuo progetto e apri il terminale nella cartella appena creata.
3. Inizializza il progetto Node.js: Utilizza il comando `npm init` per inizializzare il tuo progetto Node.js. Segui le istruzioni nel terminale per fornire i dettagli del progetto.
4. Installa Express.js: Utilizza il comando `npm install express` per installare il framework Express.js nel tuo progetto.
5. Crea un file di app.js o server.js: Crea un file JavaScript nel tuo progetto (ad esempio, app.js o server.js) e importa il modulo Express.js all'interno di esso utilizzando il seguente codice:

```
const express = require('express');
const app = express();
```

6. Definisci le rotte API: Utilizza il metodo `app.get()`, `app.post()`, `app.put()`, ecc. per definire le rotte API e le relative gestioni delle richieste. Ad esempio:

```
app.get('/api/users', (req, res) => {
  // Logica per gestire la richiesta API "/api/users"
  res.send('Risposta API per /api/users');
});
```

7. Avvia il server: Utilizza il metodo `app.listen()` per avviare il server e specifica la porta su cui desideri ascoltare le richieste. Ad esempio:

```
app.listen(3000, () => {
  console.log('Server avviato sulla porta 3000');
});
```

8. Esegui il server: Esegui il tuo server Node.js utilizzando il comando `node app.js` o `node server.js` nel terminale.