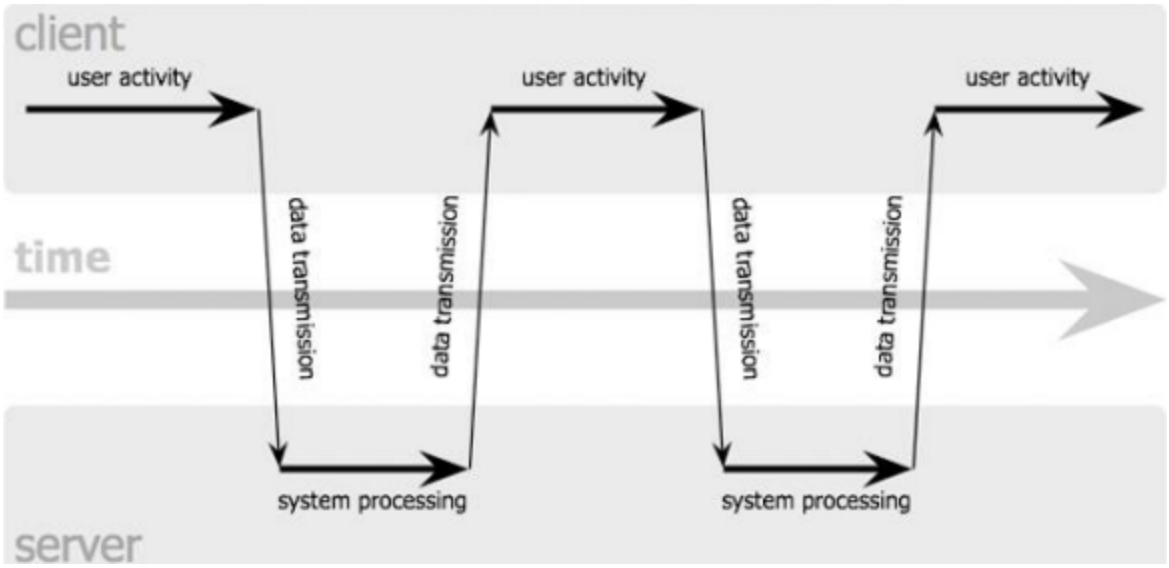
Ajax

E’ una tecnologia JS per favorire l’interazione Client Server e il refresh della pagina:

Il modello di interazione classico è molto rigido, del tipo “*click, wait and refresh*”

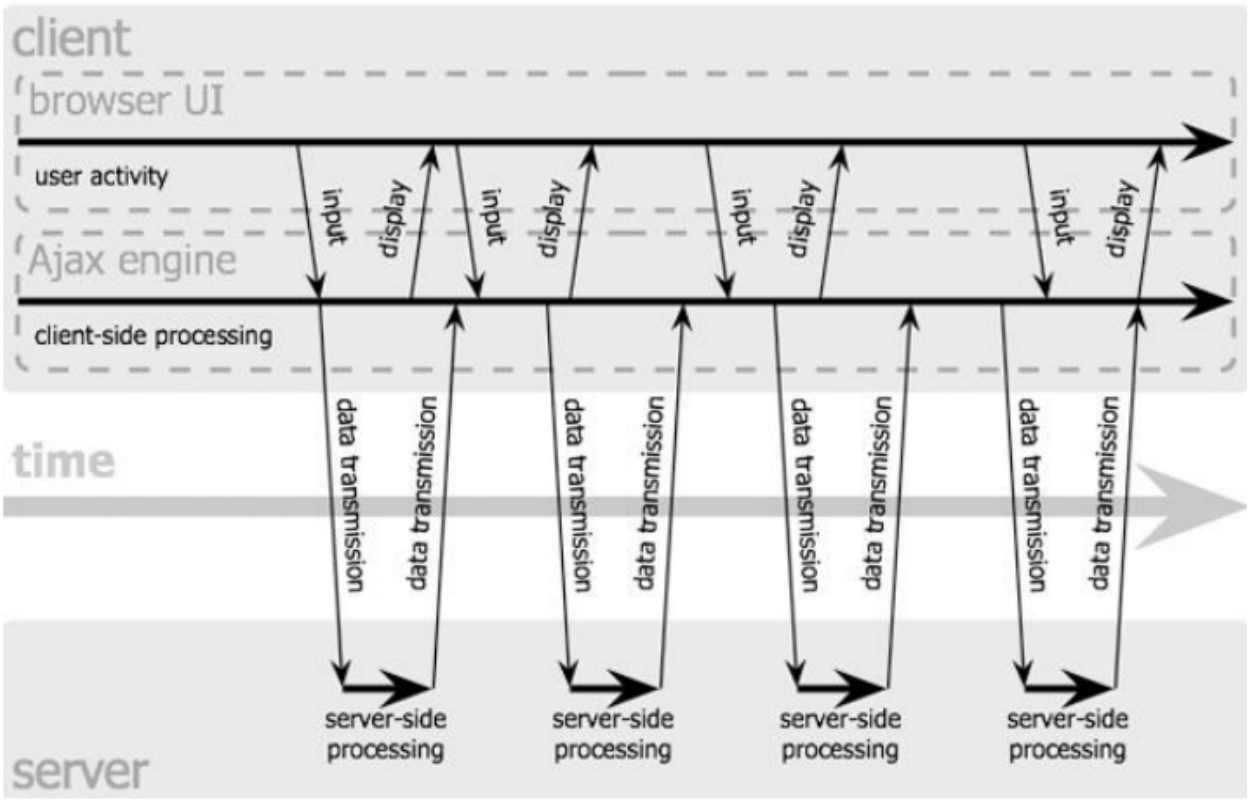


Il modello di interazione **AJAX** è nato per superare queste limitazioni.

E’ una piccola estensione di JS, basata su tecnologie standard combinate insieme per realizzare un modello di iterazione più ricco.

L’elemento centrale è l’utilizzo dell’oggetto JavaScript **XMLHttpRequest** che :

- consente di ottenere dati dal server senza necessità di ricaricare l’intera pagina.
- realizza comunicazione asincrona tra client e server: *il client non interrompe la comunicazione con l’utente anche quando è in attesa di risposte dal server.*



Tipica sequenza AJAX

Si verifica un evento determinato dall'interazione fra utente e pagine Web. L'evento comporta l'esecuzione di una funziona JS in cui:

1. si istanzia un oggetto di classe XMLHttpRequest
2. lo si configura
3. si chiama in modo asincrono il server, che elebora la richiesta e risponde al client
4. il browser invoca la funzione di callback che elabora il risultato e aggiorna il DOM

AJAX è in grado di effettuare sia richieste GET e POST, e possono essere sia sincrone che asincrone (*a noi interessano queste ultime*).

```
var xhr = new XMLHttpRequest()
```

Metodi di XMLHttpRequest

- `open(method, url, async)`: ha lo scopo di iniziare la richiesta da formulare al servlet. *method* è una stringa che assume il valore get o post*, url* una stringa che specifica la locazione del file su cui lavorare*, async* un valore booleano che indica se la richiesta è di tipo asincrono o meno*.*
- `setRequestHeader(nomeHeader, valore)`: consente di impostare gli header HTTP della richiesta da inviare. Per una richiesta GET gli header sono opzionali, nelle POST necessari
- `send(body)`: consente di inviare la richiesta al server. NON è bloccante in attesa di risposta se il parametro open è stato impostato *true*. *body* è una stringa che costituisce il body della richiesta HTTP, e deve essere inserito solo se la richiesta è di tipo “post”

Lo stato e i risultati della richiesta vengono memorizzati dall’interprete JS all’interno dell’oggetto XMLHttpRequest durante la sua esecuzione.

Tra le varie proprietà troviamo:

▼ **readyState**

lettura di tipo intero che consente di leggere in ogni momento lo stato della richiesta:

- 0 (*uninitialized*): l’oggetto esiste ma non è stata ancora chiamata la `open`
- 1 (*open*): è stato invocato il metodo `open` ma la `send` non ha ancora effettuato l’invio dei dati
- 2 (*sent*): il metodo `send` è stato eseguito e ha effettuato la richiesta
- 3 (*receiving*): la risposta viene processata

- 4 (*loaded*): l'operazione è stata completata

▼ **onreadystatechange**

definisce una funzione da chiamare quando la proprietà **readyState** cambia: infatti l'esecuzione del codice non si blocca sulla `send` in attesa dei risultati, e per gestire la risposta si deve quindi adottare un approccio a eventi. Occorre registrare una funzione di callback che viene chiamata in modo asincrono ad ogni cambio di stato della proprietà readyState. *l'assegnamento va ovviamente fatto prima della* `send`

```
xhr.onReadyStateChange = nomefunzione
```

▼ **status**

contiene il codice HTTP dell'esito della richiesta, ad esempio 200: "ok", 403="forbidden", 404="not found" etc.

▼ **statusText**

contiene una descrizione testuale del codice HTTP restituito dal server

▼ **responseText**

stringa che contiene il body della risposta HTTP (*disponibile solo che readyState == 4*)

▼ **responseXML**

body della risposta convertito in documento XML

Examples

- Get Request:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhr.send();
```

- Post Request:

```
var xhr = new XMLHttpRequest();
xhr.open("POST", "demo_post2.asp", true);
xh.setRequestHeader("Content-type", "application/x-www-form-urle
xhr.send("fname=Henry&lname=Ford");            //passaggio para
```

- Get with onreadystatechange:

```
xhr.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.queryRequest("demo").innerHTML = this.responseText
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

JSON

JSON è l'acronimo di JavaScript Object Notation. E' un formato per lo scambio di dati, considerato molto più comodo di XML: è leggero in termini di quantità di dati scambiati, efficacie e semplice da leggere "normalmente".

La sintassi JSON si basa su quella delle costanti oggetto e array di JS. Un "oggetto JSON" altro non è che una stringa equivalente a una costante oggetto di JS:

```
{"nome" : "riccardo",
 "cognome" : "palumbo",
 "età" : 20,
}
```

JS mette a disposizione la funzione `eval()` che invoca l'interprete per la traduzione della stringa passata come parametro:

- con `eval` possiamo trasformare una stringa JSON in un oggetto

- la sintassi della stringa passata a `eval` deve essere racchiusa tra parente tonde

`eval` però è rischioso: la stringa passata come parametro potrebbe essere del codice malevolo, perciò si preferisce utilizzare parser appositi che traducono solo oggetti JSON e non espressioni JS di qualunque tipo.

GSON

GSON è una libreria standard per il parsing/deparsing di oggetti JSON.

```
Gson g = new Gson();

//serializzazione di un oggetto
Person santa = new Person("Santa", "Claus", 100);
String santaJson = g.toJson(santa);

//deserializzazione di un oggetto
Person peterPan = g.fromJson(json, Person.class);
```

JSON E AJAX

In un'interazione client-server in cui il cliente vuole trasferire un oggetto JS.

Sul lato client:

- si crea un nuovo oggeto JS e lo si riempie con le informazioni da noi volute.
- si usa `JSON.stringify()` per convertite l'oggetto in stringa JSON
- si una la funzione `encodeURIComponent()` per convertire la stringa in un formato utilizzabile in una richiesta HTTP
- si manda la stringa al server mediante `XMLHttpRequest`. *che viene passata come variabili con GET o POST*)

Sul lato server:

- si decodifica la stringa JSON e la si trasforma in un oggetto Java utilizzando l’apposito parser.
- si elabora l’oggetto
- si crea un nuovo oggetto Java che contiene dati della risposta
- si trasforma l’oggetto Java in stringa JSON usando il parser suddetto
- si trasmette la stringa JSON al client nel corpo della risposta HTTP

Sul lato client, all’atto della ricezione:

- si converte la stringa JSON in un oggetto JS usanto `JSON.parse()`
- si usa liberatamente l’oggetto per gli scopi desiderati