


Développement Web J2EE :
- HTTP, Servlet, JSP, MVC

LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion

LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion

Serveur Web

LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion

Client HTTP

Serveur Web

```
:ServerSocket  
port=80  
accept()
```

LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP...)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion

Client HTTP

:Socket

IPS=
Port=80

Serveur Web

:ServerSocket

port=80
accept()

Doc.htm

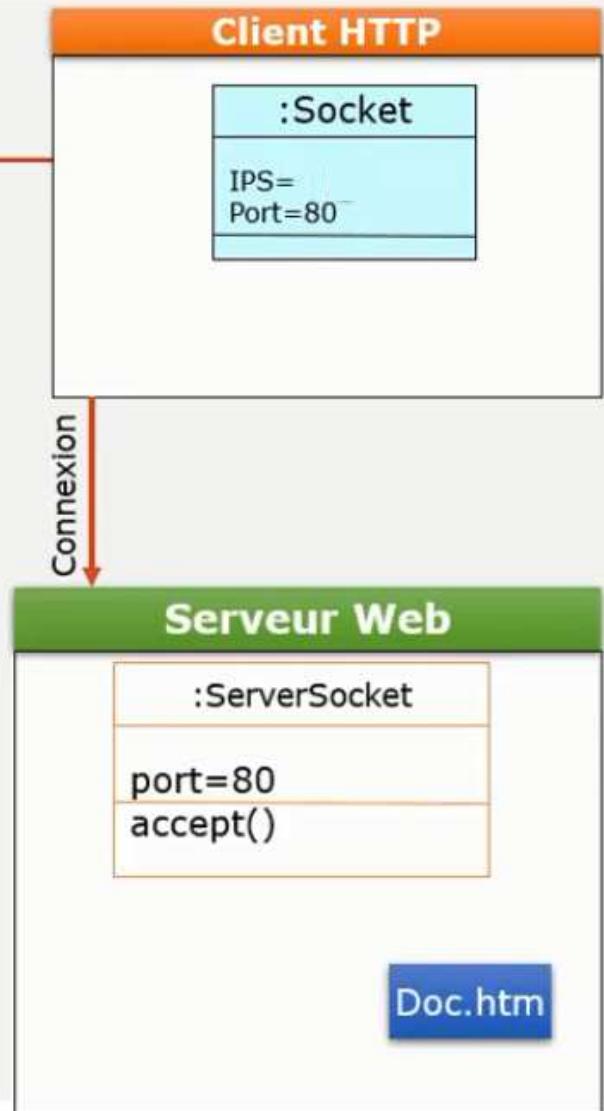
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP...)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



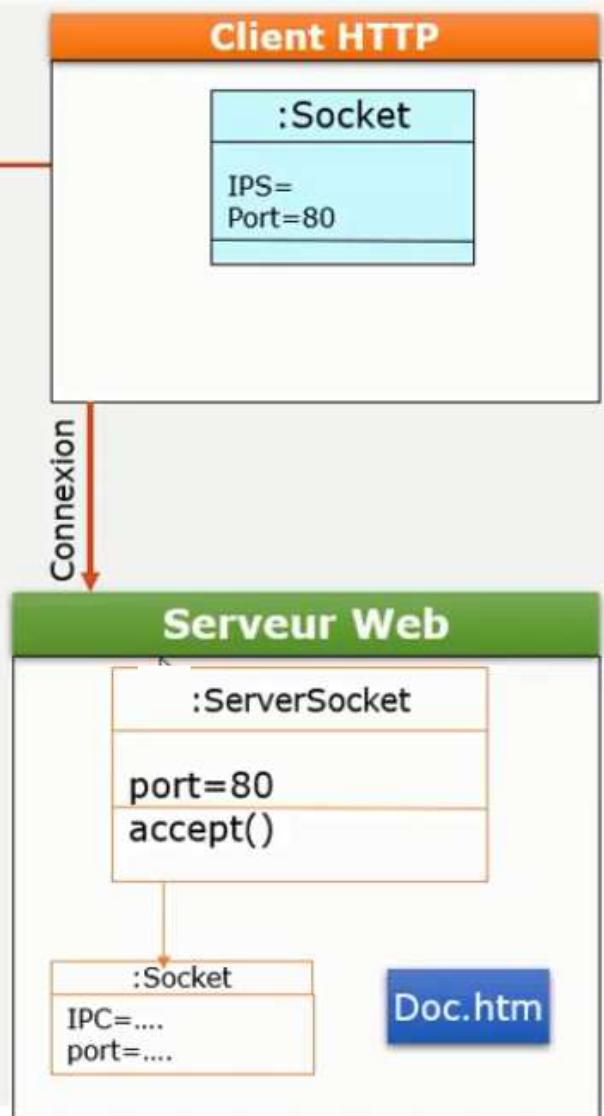
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



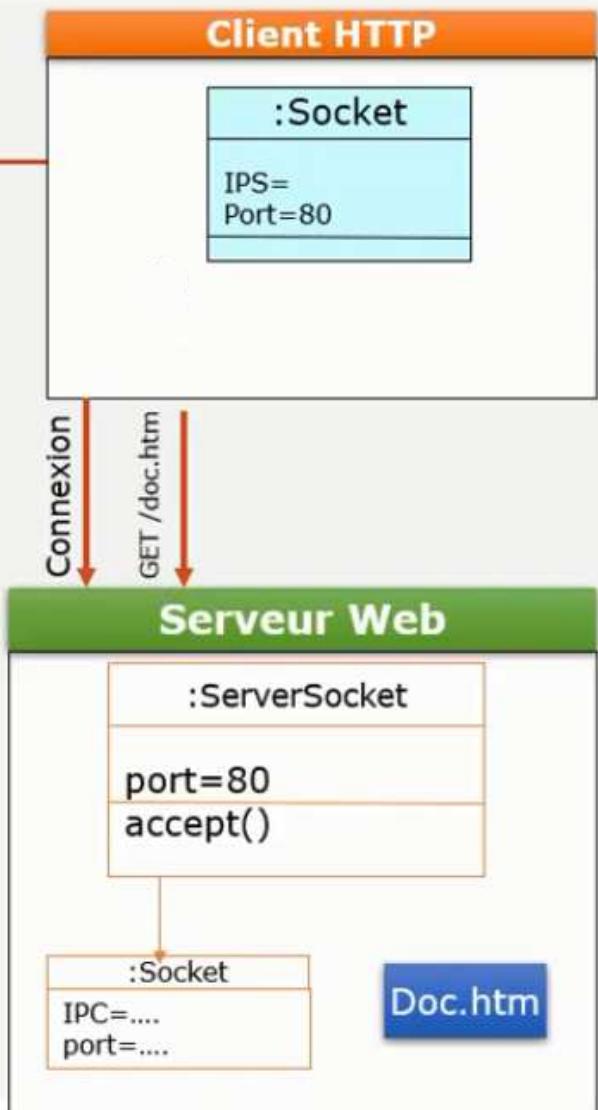
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP...)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



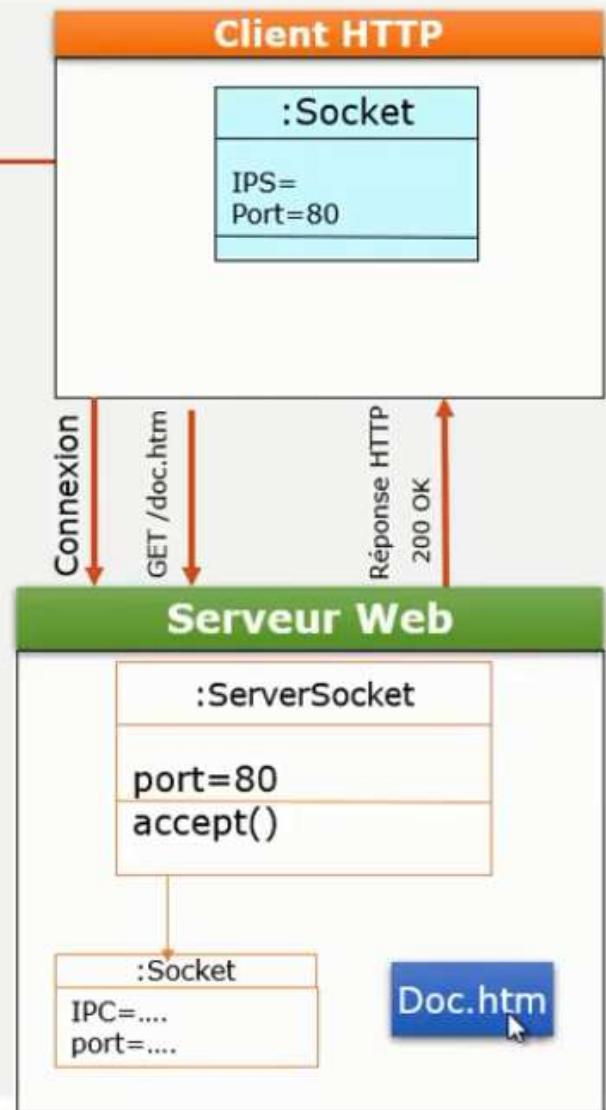
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



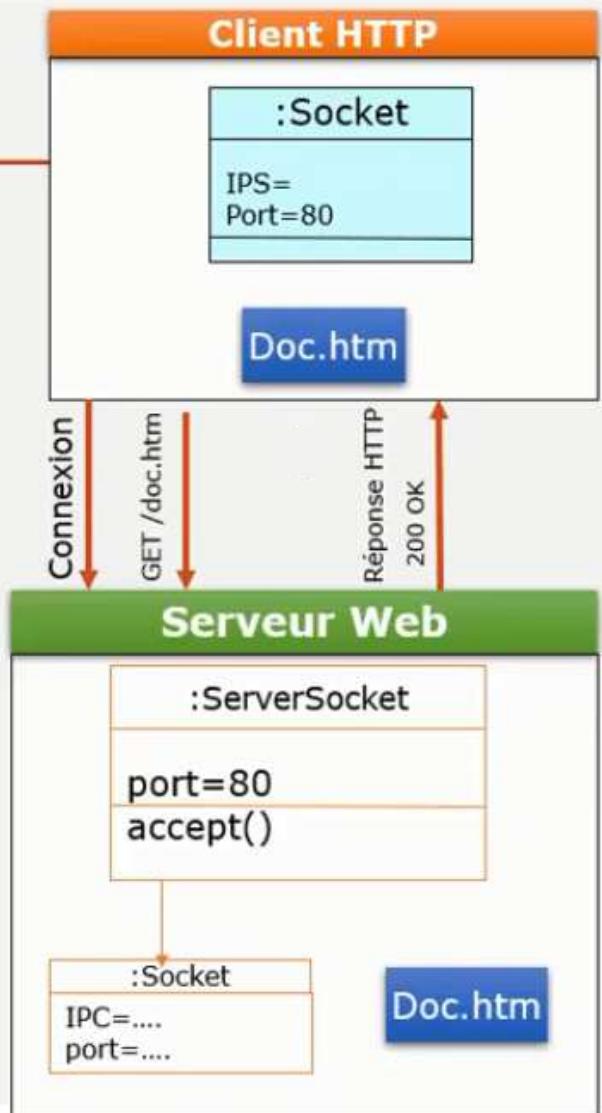
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP...)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



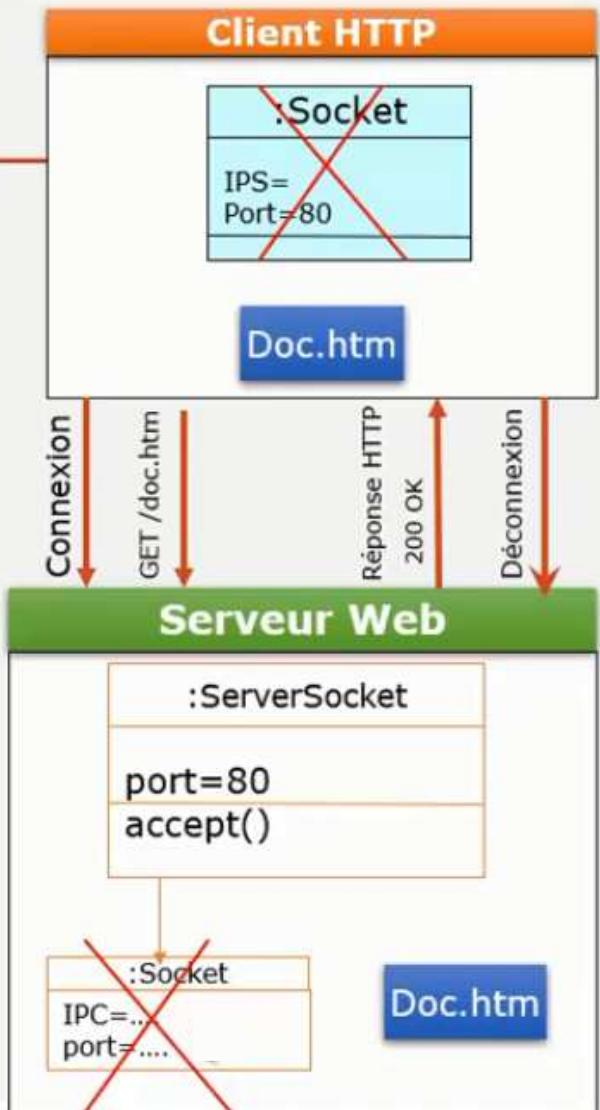
LE PROTOCOLE HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP..)
- Ce protocole permet également de soumissionner les formulaires

Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- Déconnexion



LE PROTOCOLE HTTP

- **Etapes d'une requête HTTP**

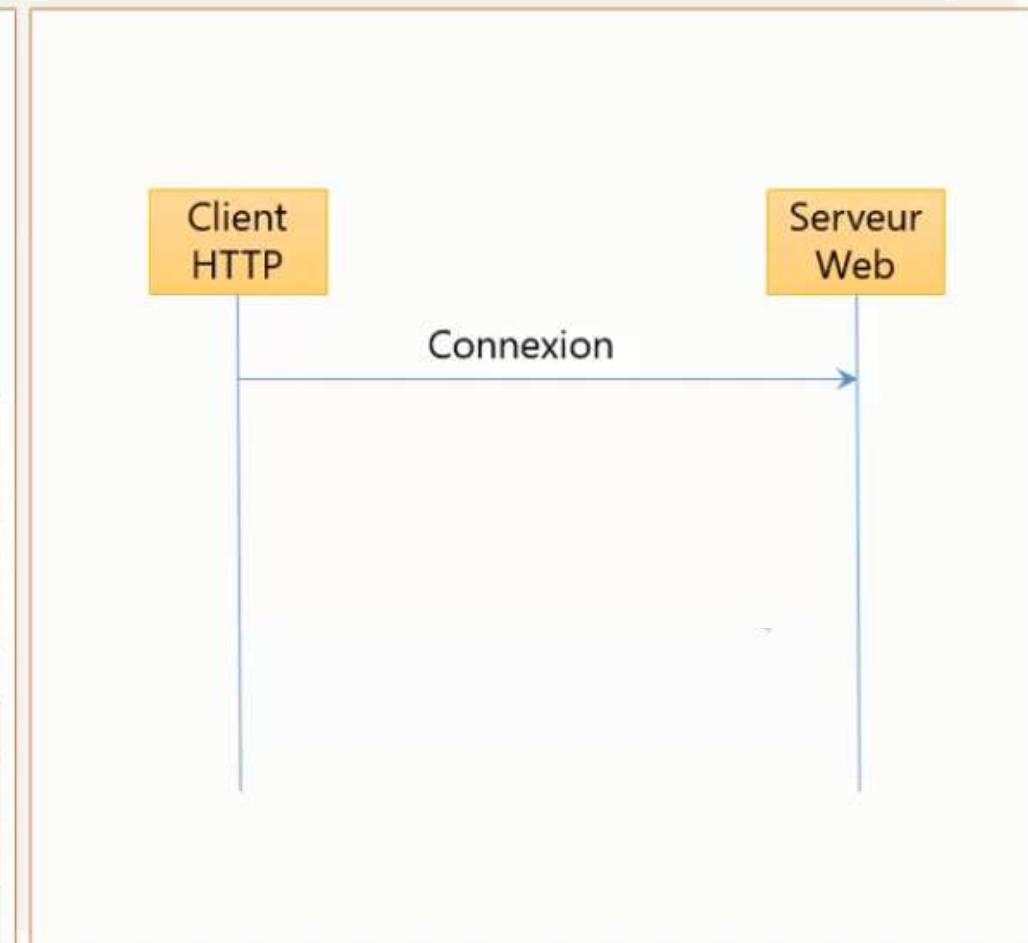
1. Le client se connecte au serveur (Créer une socket)
2. Le client demande au serveur un document : Requête HTTP
3. Le serveur renvoie au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
4. Déconnexion

- **Structure d'une requête HTTP :**

1. Méthodes /ressource http/1.1 → POST /login http/1.1
2. Entêtes de la requête HTTP → Accept : application/json)
3. Saut de ligne →
4. Corps de la requête HTTP → username=x&password=xxx

- **Structure d'une réponse HTTP :**

1. http/1.1 Status_Code Status_Text → http/1.1 200 OK
2. Entêtes de la réponse HTTP → Content-Type : text/html)
3. Saut de ligne →
4. Corps de la réponse HTTP → <html><body></body></html>)



LE PROTOCOLE HTTP

- **Etapes d'une requête HTTP**

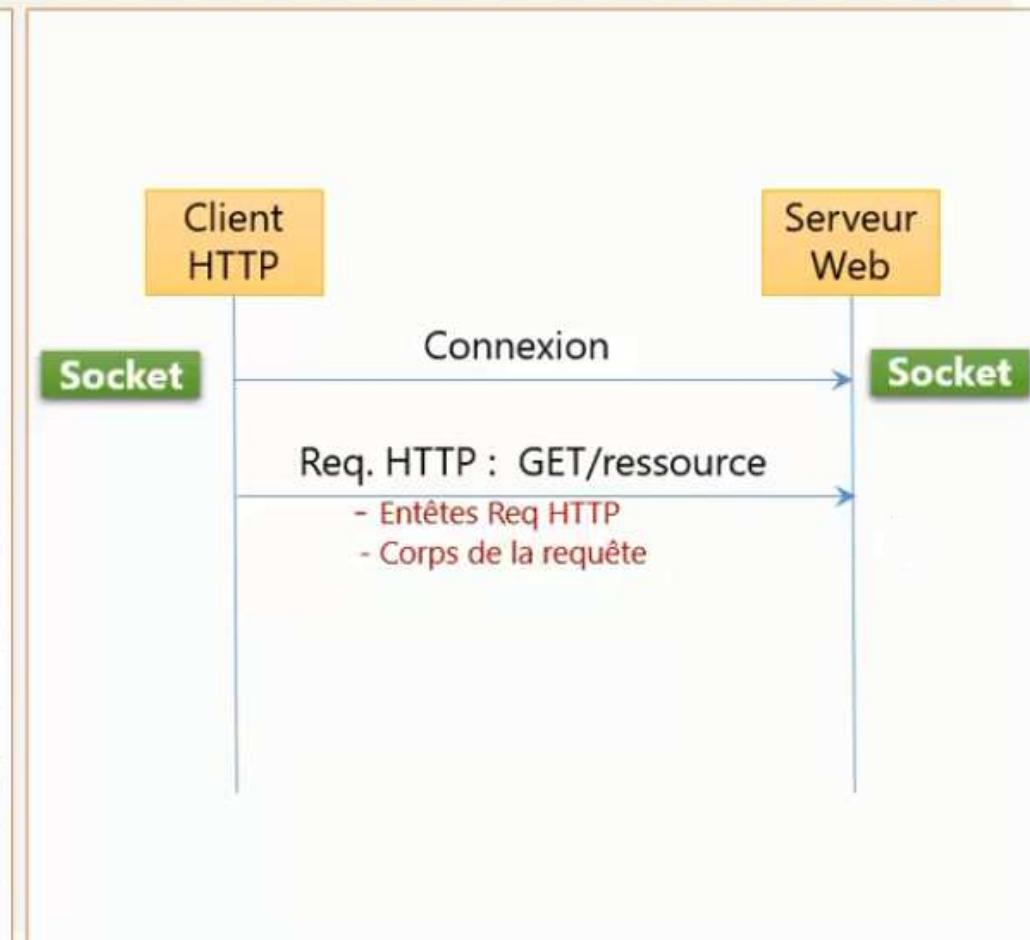
1. Le client se connecte au serveur (Créer une socket)
2. Le client demande au serveur un document : Requête HTTP
3. Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
4. Déconnexion

- **Structure d'une requête HTTP :**

1. Méthodes /ressource http/1.1 → POST /login http/1.1)
2. Entêtes de la requête HTTP → Accept : application/json)
3. Saut de ligne →
4. Corps de la requête HTTP → username=x&password=xxx

- **Structure d'une réponse HTTP :**

1. http/1.1 Status_Code Status_Text → http/1.1 200 OK)
2. Entêtes de la réponse HTTP → Content-Type : text/html)
3. Saut de ligne →
4. Corps de la réponse HTTP → <html><body></body></html>)



LE PROTOCOLE HTTP

- **Etapes d'une requête HTTP**

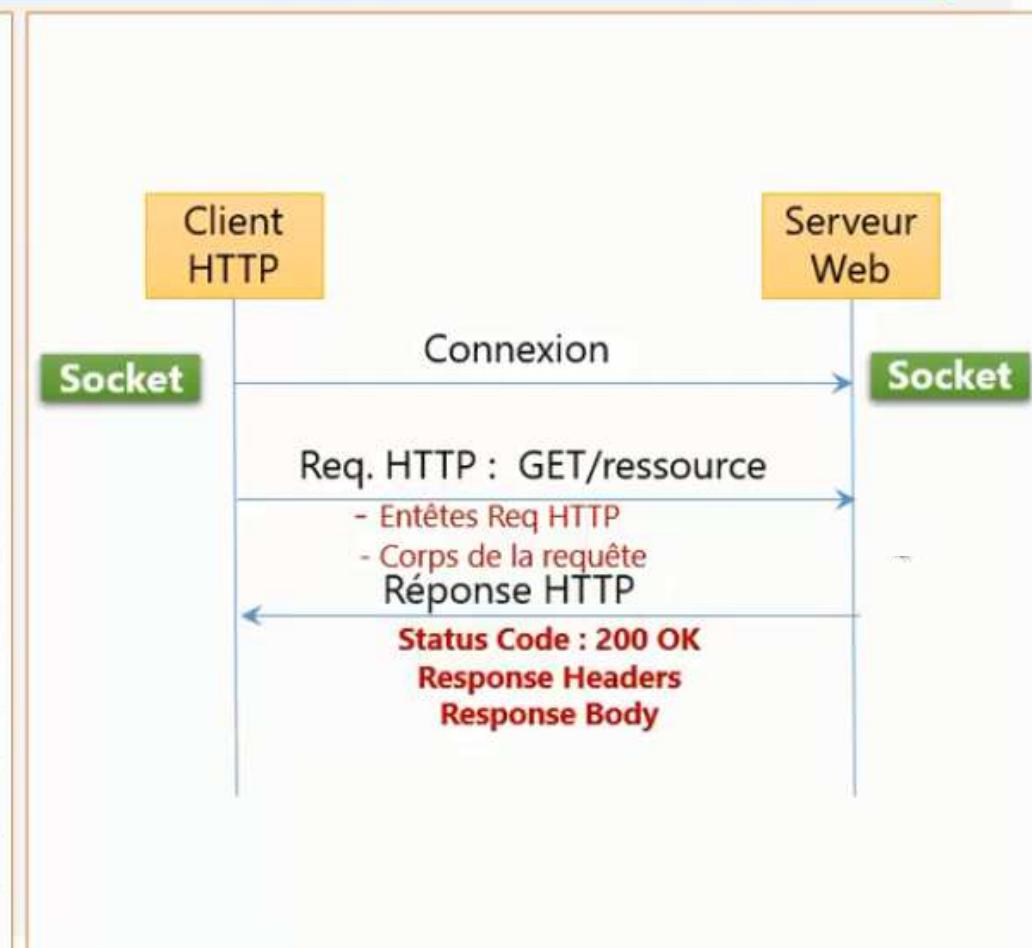
1. Le client se connecte au serveur (Créer une socket)
2. Le client demande au serveur un document : Requête HTTP
3. Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
4. Déconnexion

- **Structure d'une requête HTTP :**

1. Méthodes /ressource http/1.1 → POST /login http/1.1)
2. Entêtes de la requête HTTP → Accept : application/json)
3. Saut de ligne →
4. Corps de la requête HTTP → username=x&password=xxx

- **Structure d'une réponse HTTP :**

1. http/1.1 Status_Code Status_Text → http/1.1 200 OK
2. Entêtes de la réponse HTTP → Content-Type : text/html)
3. Saut de ligne →
4. Corps de la réponse HTTP → <html><body></body></html>)



LE PROTOCOLE HTTP

- **Etapes d'une requête HTTP**

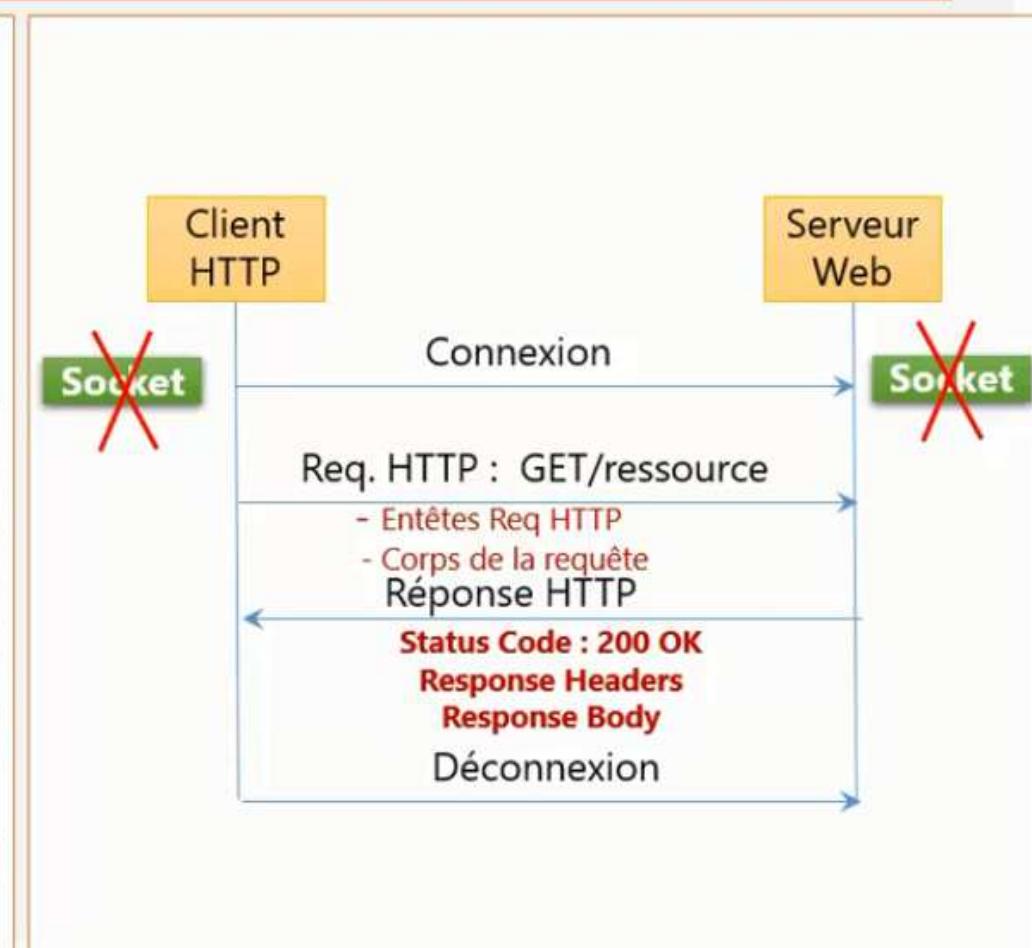
1. Le client se connecte au serveur (Créer une socket)
2. Le client demande au serveur un document : Requête HTTP
3. Le serveur renvoie au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
4. Déconnexion

- **Structure d'une requête HTTP :**

1. Méthodes /ressource http/1.1 → POST /login http/1.1
2. Entêtes de la requête HTTP → Accept: application/json)
3. Saut de ligne →
4. Corps de la requête HTTP → username=x&password=xxx

- **Structure d'une réponse HTTP :**

1. http/1.1 Status_Code Status_Text → http/1.1 200 OK
2. Entêtes de la réponse HTTP → Content-Type : text/html)
3. Saut de ligne →
4. Corps de la réponse HTTP → <html><body></body></html>)

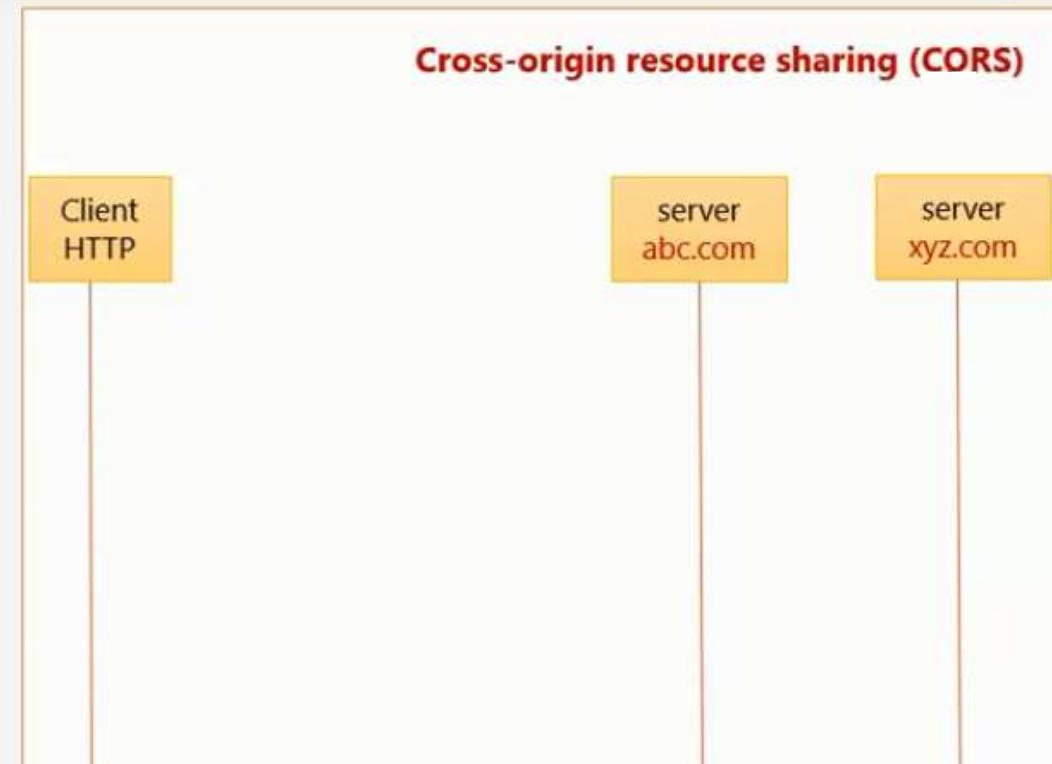


Méthodes du protocole HTTP

Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

1. **GET** : Pour Consulter des ressources
2. **POST** : Pour soumissionner et ajouter de nouvelles ressources
3. **PUT** pour mettre à jour des ressources existantes
4. **DELETE** pour supprimer des ressources existantes.
5. **HEAD** permet de consulter les métadonnées d'une ressource (Type, Capacité, Date de dernière modification etc...)
6. **OPTIONS** : Permet au client HTTP d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisées, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).

Cross-origin resource sharing (CORS)



Méthodes du protocole HTTP

Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

1. **GET** : Pour Consulter des ressources
2. **POST** : Pour soumissionner et ajouter de nouvelles ressources
3. **PUT** pour mettre à jour des ressources existantes
4. **DELETE** pour supprimer des ressources existantes.
5. **HEAD** permet de consulter les métadonnées d'une ressource (Type, Capacité, Date de dernière modification etc...)
6. **OPTIONS** : Permet au client HTTP d'intéroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).

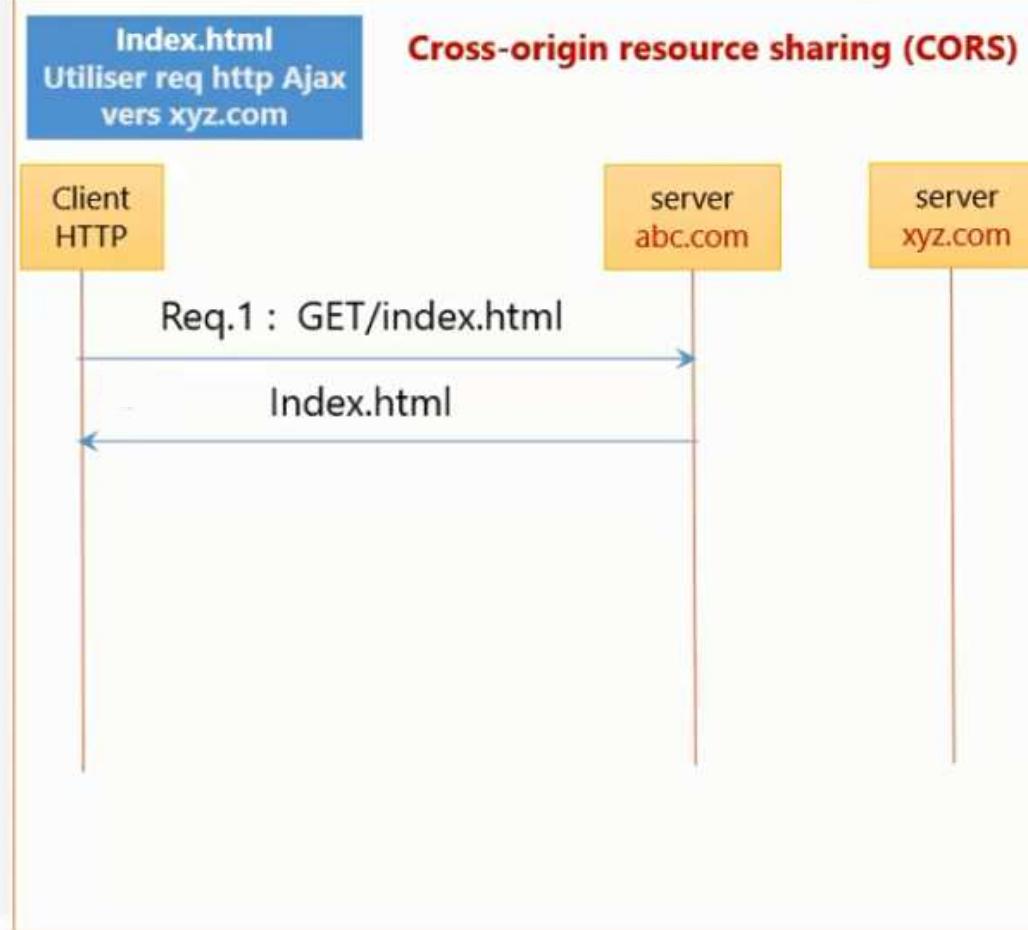
Cross-origin resource sharing (CORS)



Méthodes du protocole HTTP

Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

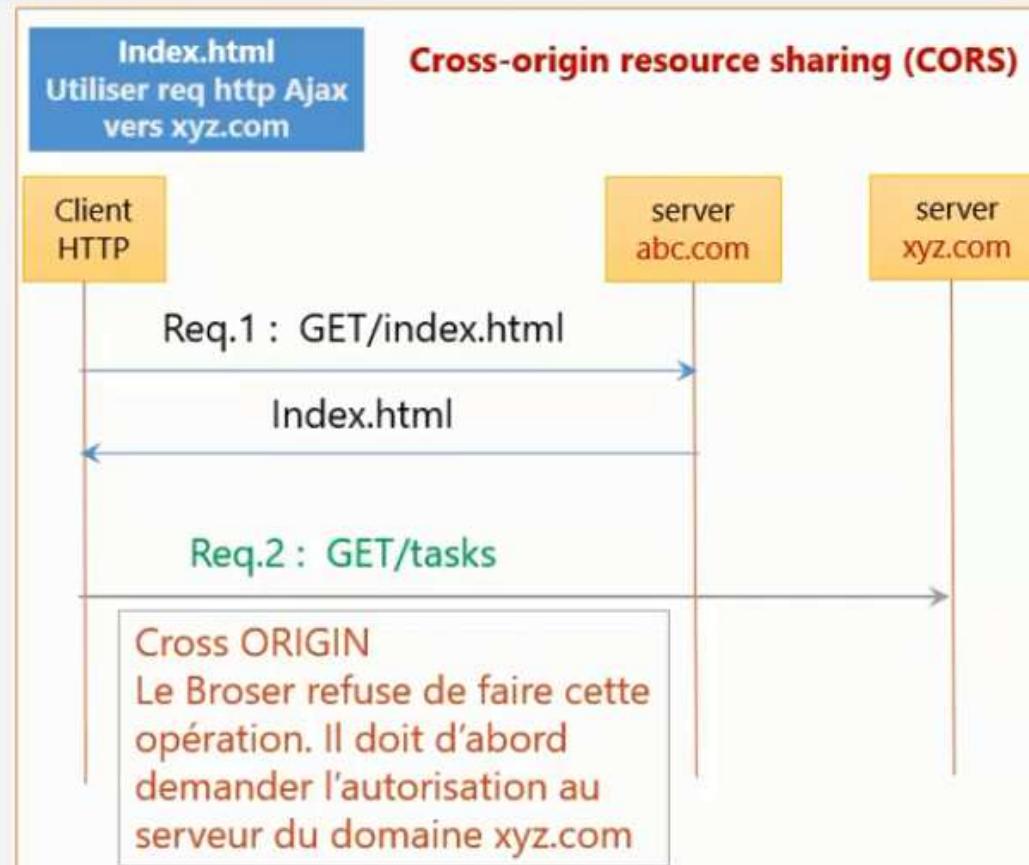
1. **GET** : Pour Consulter des ressources
2. **POST** : Pour soumissionner et ajouter de nouvelles ressources
3. **PUT** pour mettre à jour des ressources existantes
4. **DELETE** pour supprimer des ressources existantes.
5. **HEAD** permet de consulter les métadonnées d'une ressource (Type, Capacité, Date de dernière modification etc...)
6. **OPTIONS** : Permet au client HTTP d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisées, etc...). Utilisé souvent comme requête de pré vérification Cross-Origin CORS (Cross Origin Resource Sharing).



Méthodes du protocole HTTP

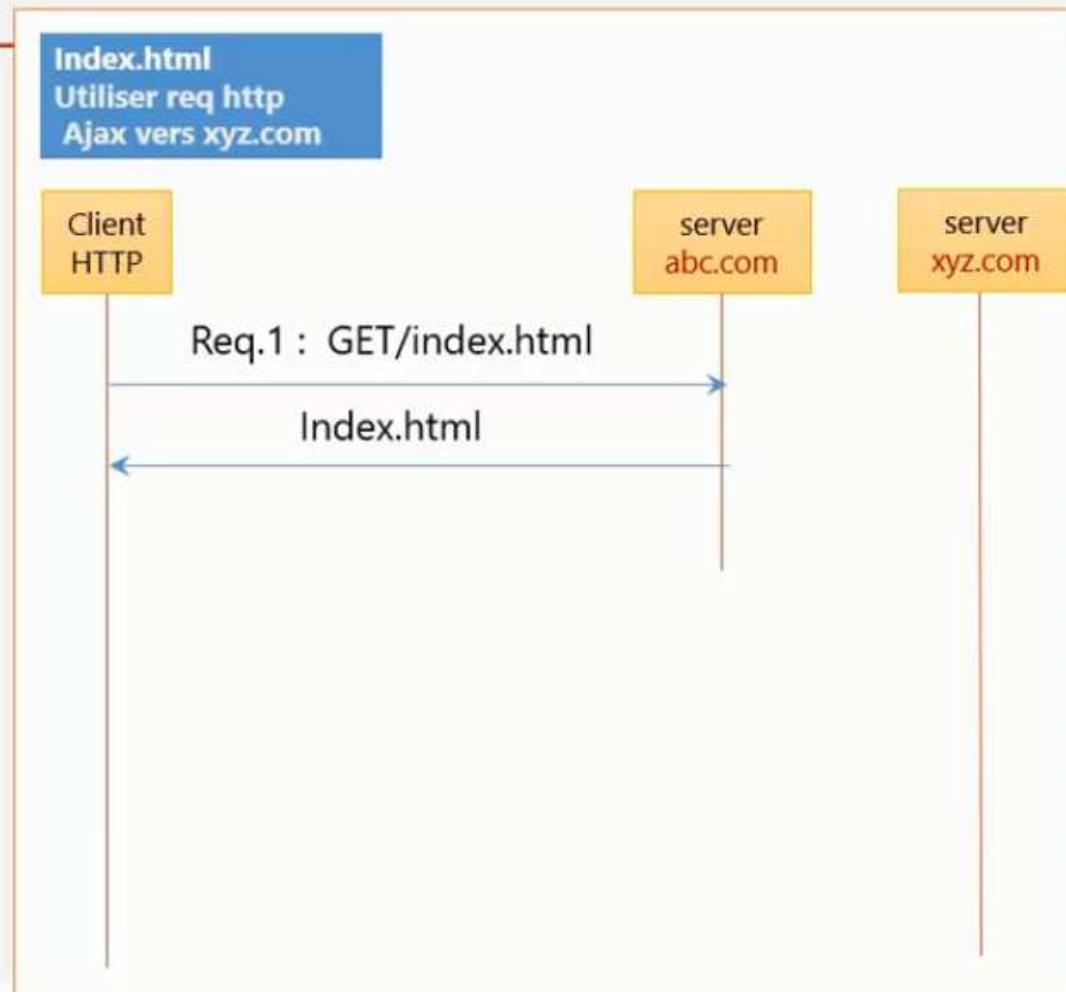
Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

1. **GET** : Pour Consulter des ressources
2. **POST** : Pour soumissionner et ajouter de nouvelles ressources
3. **PUT** pour mettre à jour des ressources existantes
4. **DELETE** pour supprimer des ressources existantes.
5. **HEAD** permet de consulter les métadonnées d'une ressource (Type, Capacité, Date de dernière modification etc...)
6. **OPTIONS** : Permet au client HTTP d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisées, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



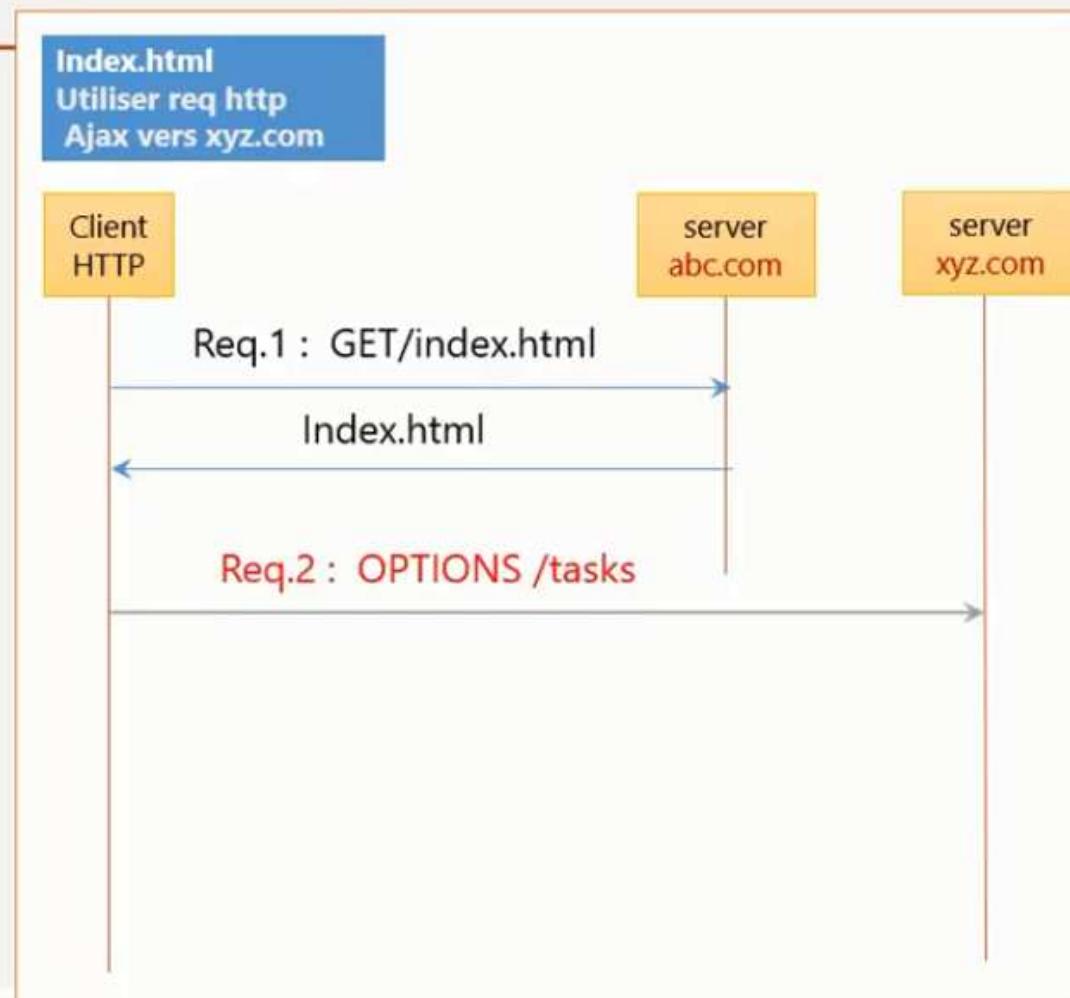
Cross-Origin Resource Sharing : CORS

- Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
- Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.
- Pour demander l'autorisation, le browser envoie d'abord une requête http avec la méthode OPTIONS Permet d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



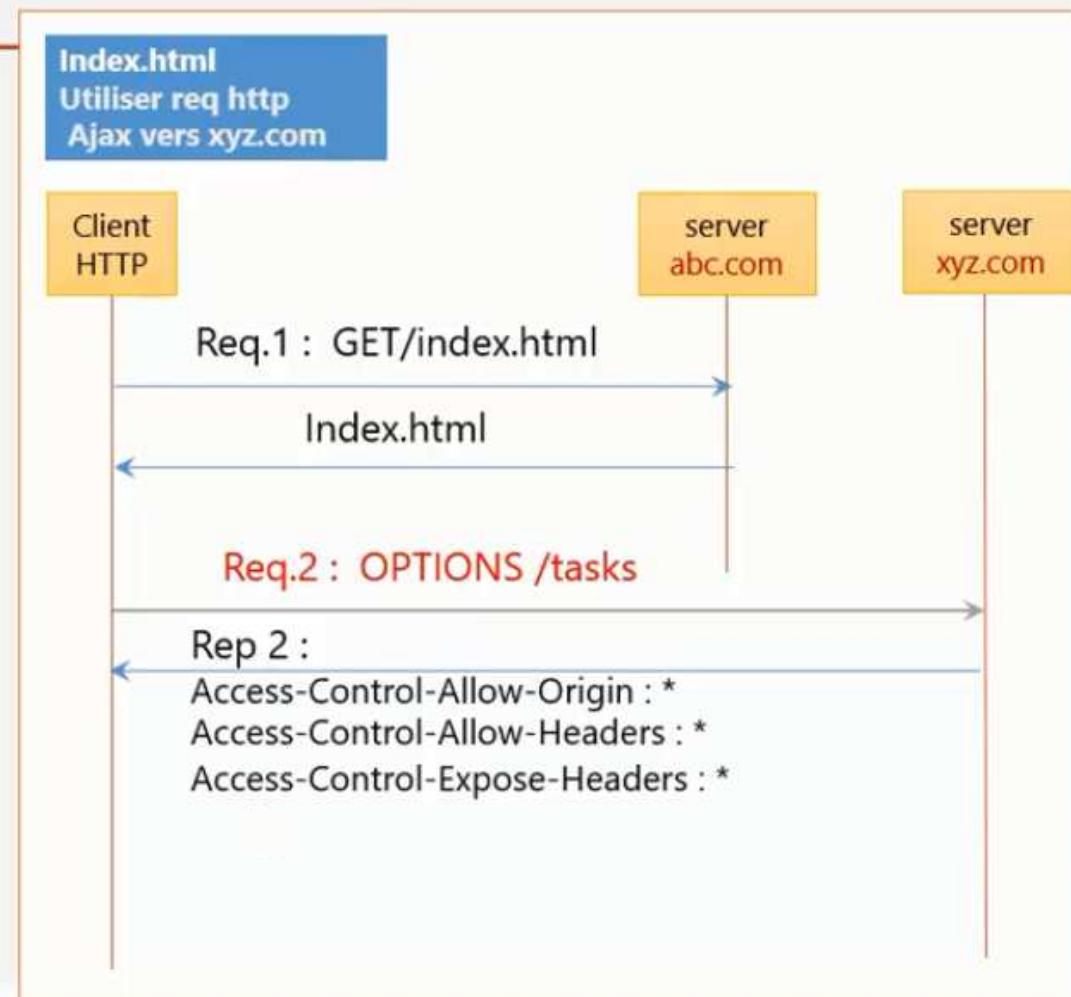
Cross-Origin Resource Sharing : CORS

- Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
- Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.
- Pour demander l'autorisation, le browser envoie d'abord une requête http avec la méthode OPTIONS Permet d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



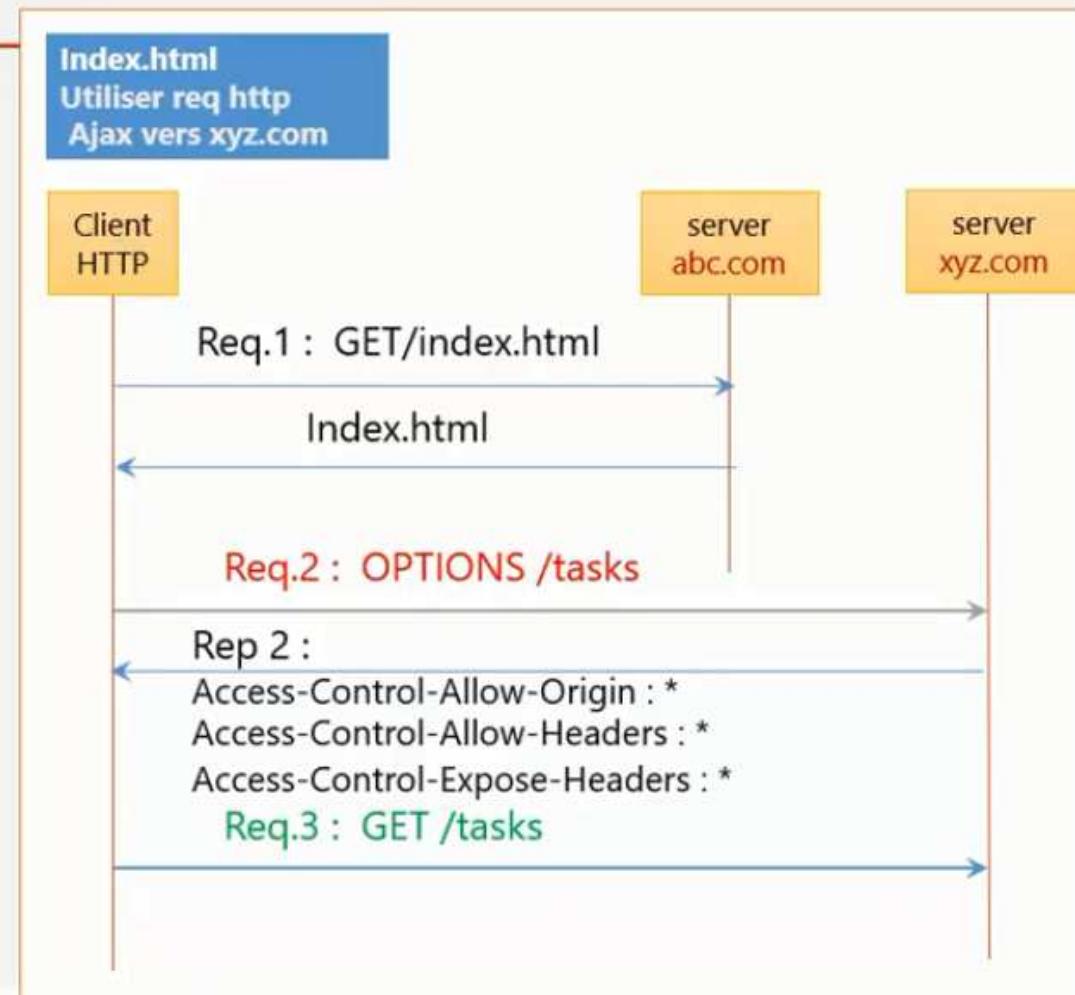
Cross-Origin Resource Sharing : CORS

- Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
- Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.
- Pour demander l'autorisation, le browser envoie d'abord une requête http avec la méthode OPTIONS Permet d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



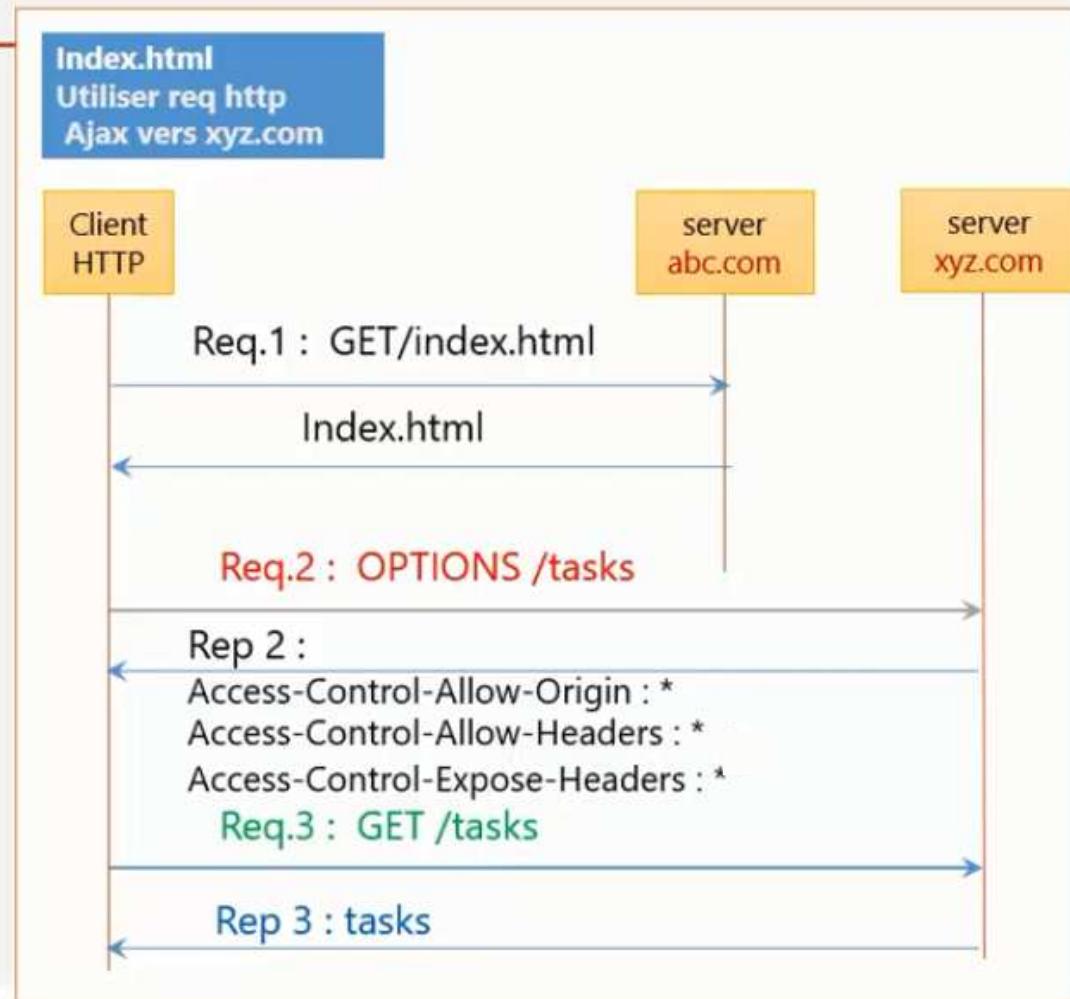
Cross-Origin Resource Sharing : CORS

- Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
- Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.
- Pour demander l'autorisation, le browser envoie d'abord une requête http avec la méthode OPTIONS Permet d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



Cross-Origin Resource Sharing : CORS

- Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un agent utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.
- Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.
- Pour demander l'autorisation, le browser envoie d'abord une requête http avec la méthode OPTIONS Permet d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources (Méthodes autorisées, Entêtes autorisées, Origines autorisés, etc...). Utilisé souvent comme requête de pré-vérification Cross-Origin CORS (Cross Origin Resource Sharing).



Exemple de requête HTTP avec POST



Exemple de requête HTTP avec POST

Entête de la requête

Post /login HTTP/1.1

host : intra.net

Accept: application/json

Content-Type : application/json

Cookie : JSESSIONID : C4714D557A7CFD66F6AFED1DD8356835

Saut de ligne

{"username":"admin","password":"1234","action":"login" }

corps de la requête

Exemple de requête HTTP avec GET



Réponse HTTP :

Entête de la réponse

HTTP/1.1 200 OK

Date : Wed, 05Feb02 15:02:01 GMT

Server : Apache/1.3.24

Last-Modified : Wed 02Oct01 24:05:01GMT

Content-Type : text/html

Content-length : 29

Set-Cookie : JSESSIONID: C4714D557A7CFD66F6AFED1DD8356835

Saut de ligne

```
<html>
  <body>
    </body>
</html>
```

corps de la réponse

Réponse HTTP :

Entête de la réponse

HTTP/1.1 200 OK

Date : Wed, 05Feb02 15:02:01 GMT

Server : Apache/1.3.24

Last-Modified : Wed 02Oct01 24:05:01GMT

Content-Type : application/json

Content-length : 77

Set-Cookie : JSESSIONID: C4714D557A7CFD66F6AFED1DD8356835

Saut de ligne

[

{"id":1,"taskName":"T1"}, {"id":2,"taskName":"T2"},

{"id":3,"taskName":"T3"}

]

corps de la réponse



Réponse HTTP : Status Code

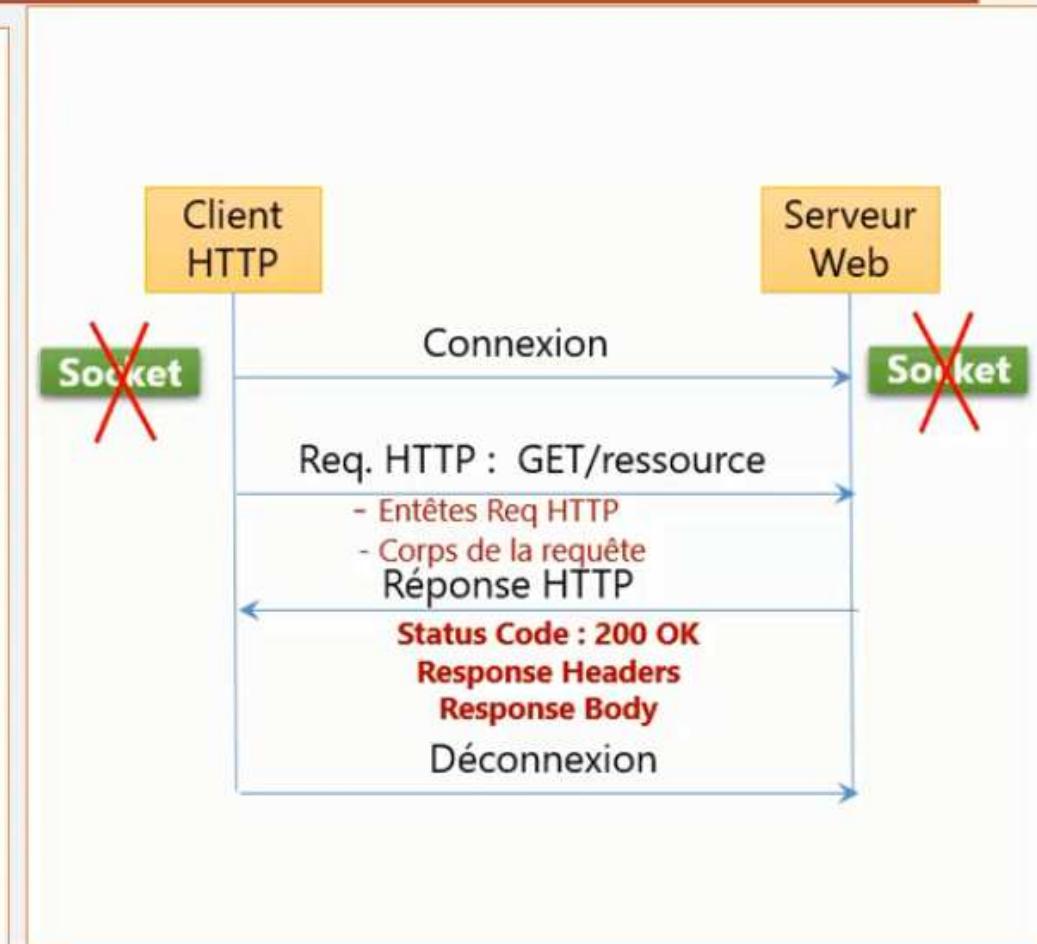
Les principales valeurs des codes de statut HTTP sont détaillées dans le tableau ci-après.

Information 1xx :

- 100 (Continue) : Utiliser dans le cas où la requête possède un corps.
- 101 (Switching protocol) : Demander au client de changer de protocole. Par exemple de Http1.0 vers Http 1.1 ou vers web socket

Succès 2xx :

- 200 (OK) : Le document a été trouvé et son contenu suit
- 201 (Created) : Le document a été créé en réponse à un PUT
- 202 (Accepted) : Requête acceptée, mais traitement non terminé
- 204 (No response) : Le serveur n'a aucune information à renvoyer
- 206 (Partial content) : Une partie du document suit



Réponse HTTP : Status Code

Redirection 3xx :

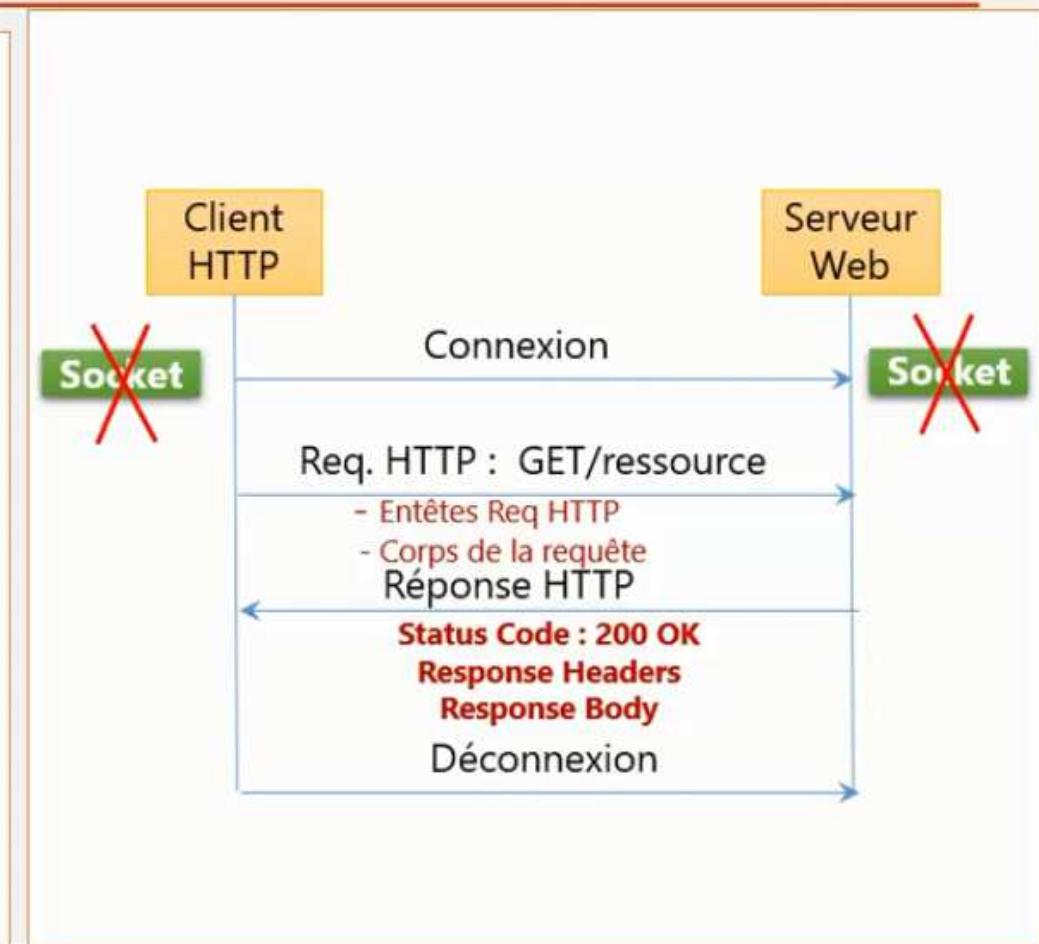
- 301 (Moved) : Le document a changé d'adresse de façon permanente
- 302 (Found) : Le document a changé d'adresse temporairement
- 304 (Not modified) : Le document demandé n'a pas été modifié

Erreurs du client 4xx :

- 400 (Bad request) : La syntaxe de la requête est incorrecte
- 401 (Unauthorized) : Le client n'a pas les privilèges d'accès au document
- 403 (Forbidden) : L'accès au document est interdit
- 404 (Not found) : Le document demandé n'a pu être trouvé
- 405 (Method not allowed) : La méthode de la requête n'est pas autorisée

Erreurs du serveur 5xx :

- 500 (Internal error) : Une erreur inattendue est survenue au niveau du serveur
- 501 (Not implemented) : La méthode utilisée n'est pas implémentée
- 502 (Bad gateway) : Erreur de serveur distant lors d'une requête proxy



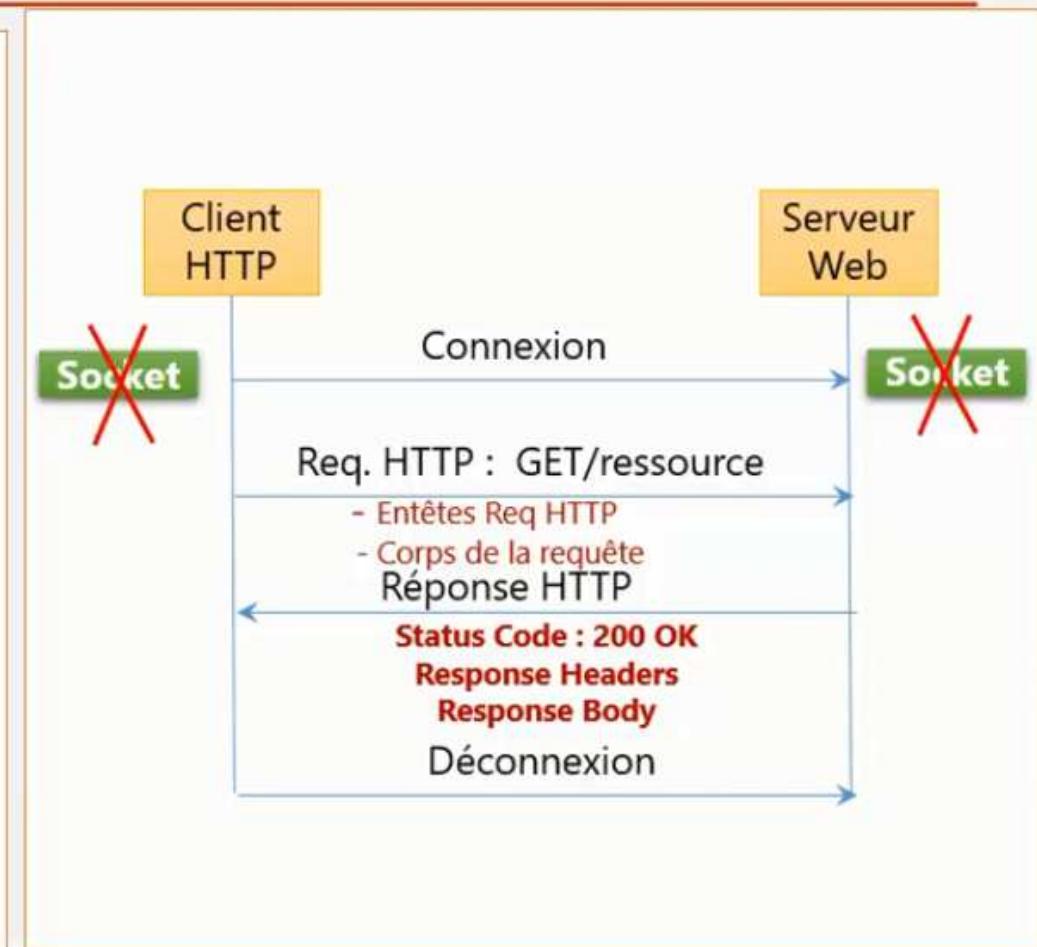
Entêtes HTTP

Entêtes HTTP génériques :

- **Content-length** : Longueur en octets des données suivant les entêtes
- **Content-type** : Type MIME des données qui suivent
- **Connection** : Indique si la connexion TCP doit rester ouverte (Keep-Alive) ou être fermée (close)

Entêtes de la requête :

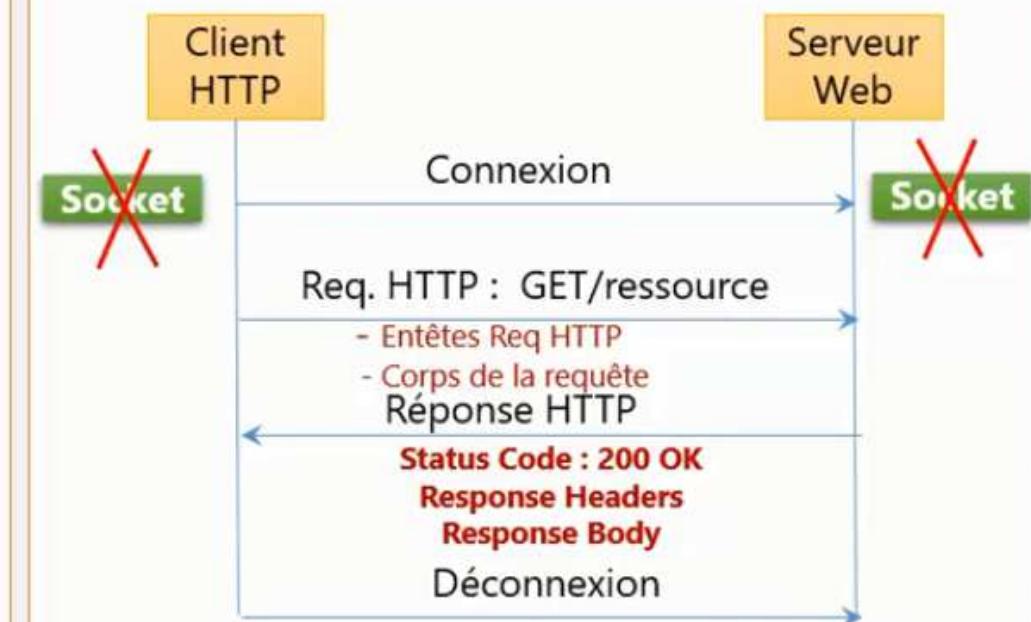
- **Accept** : Types MIME que le client accepte
- **Accept-encoding** : Méthodes de compression supportées par le client
- **Accept-language** : Langues préférées par le client (pondérées)
- **Cookie** : Données de cookie mémorisées par le client
- **Host** : Hôte virtuel demandé
- **If-modified-since** : Ne retourne le document que si modifié depuis la date indiquée
- **If-none-match** : Ne retourne le document que s'il a changé
- **Referer** : URL de la page à partir de laquelle le document est demandé
- **User-agent** : Nom et version du logiciel client



Entêtes HTTP

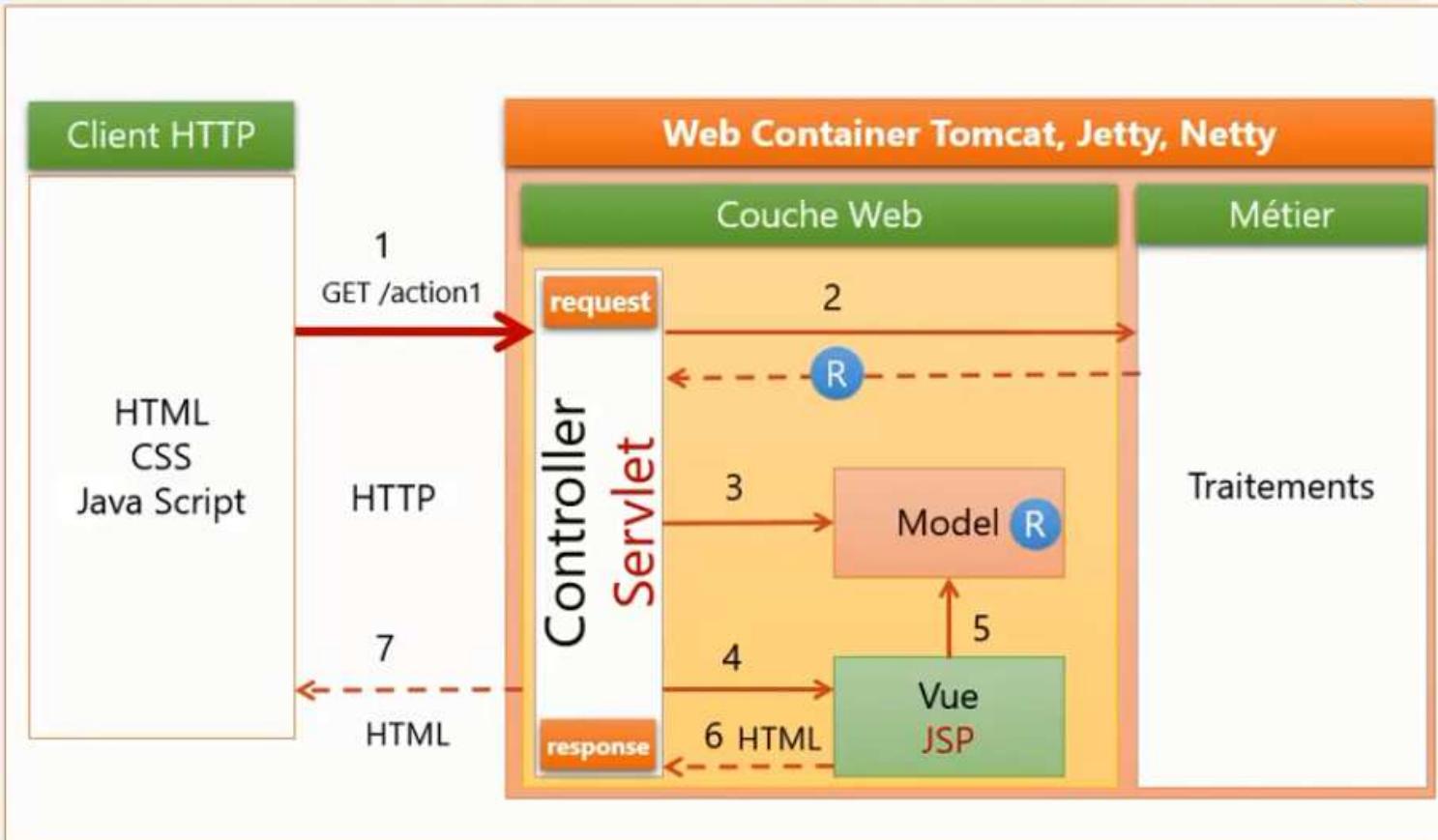
- **Entêtes de la réponse Http:**

- **Allowed** : Méthodes HTTP autorisées pour cette URI (comme POST)
- **Content-encoding** : Méthode de compression des données qui suivent
- **Content-language** : Langue dans laquelle le document retourné est écrit
- **Date** : Date et heure UTC courante
- **Expires** : Date à laquelle le document expire
- **Last-modified** : Date de dernière modification du document
- **Location** : Adresse du document lors d'une redirection
- **Etag** : Numéro de version du document
- **Pragma** : Données annexes pour le navigateur (par exemple, no.cache)
- **Server** : Nom et version du logiciel serveur
- **Set-cookie** : Permet au serveur d'écrire un cookie sur le disque du client



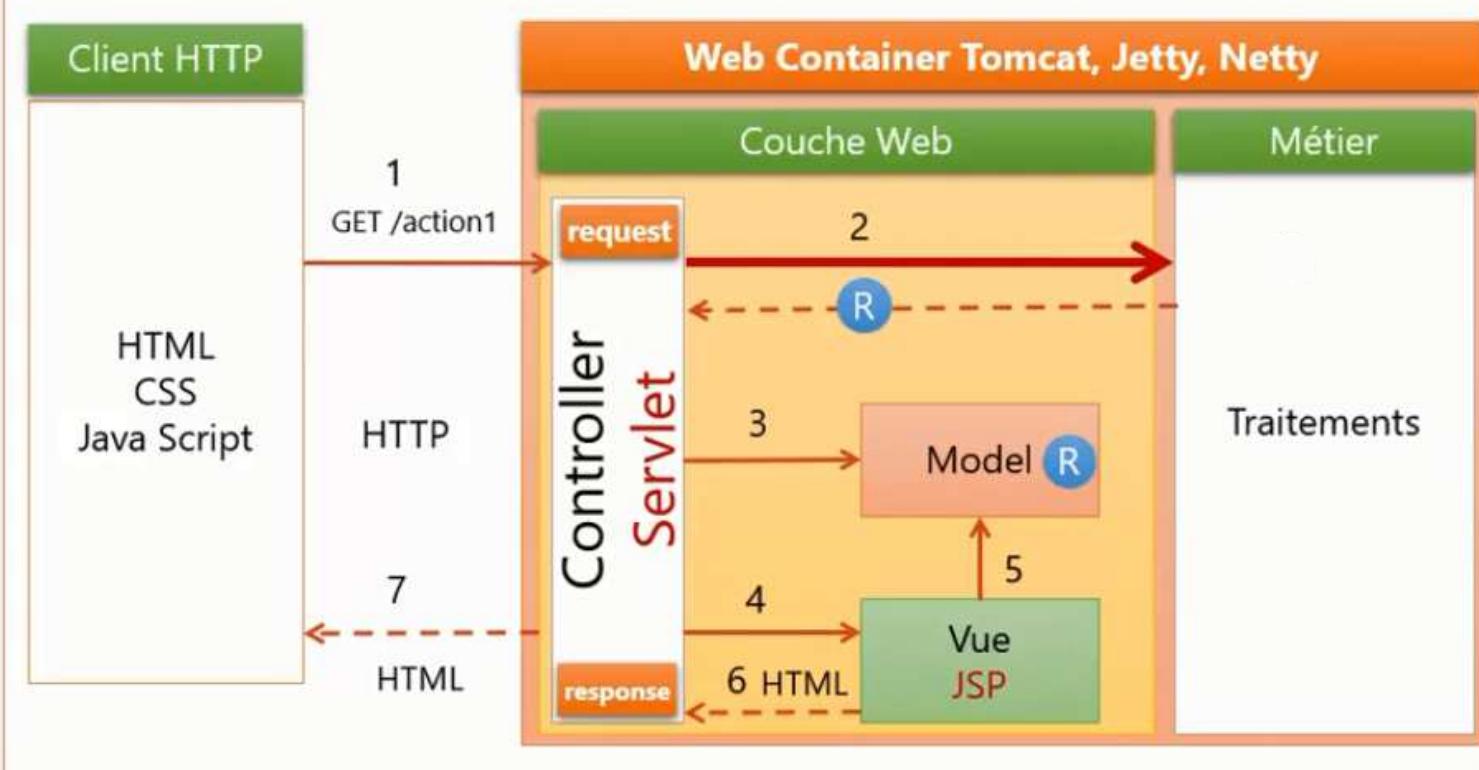
Architecture Web JEE : HTTP, Servlet, JSP, MVC

1 – Le client envoie une requête HTTP de type GET ou POST vers le contrôleur représenté par un composant Web JEE : **SERVLET**. Pour lire les données de la requête HTTP le contrôleur utilise l'objet **request** de type **HttpServletRequest**



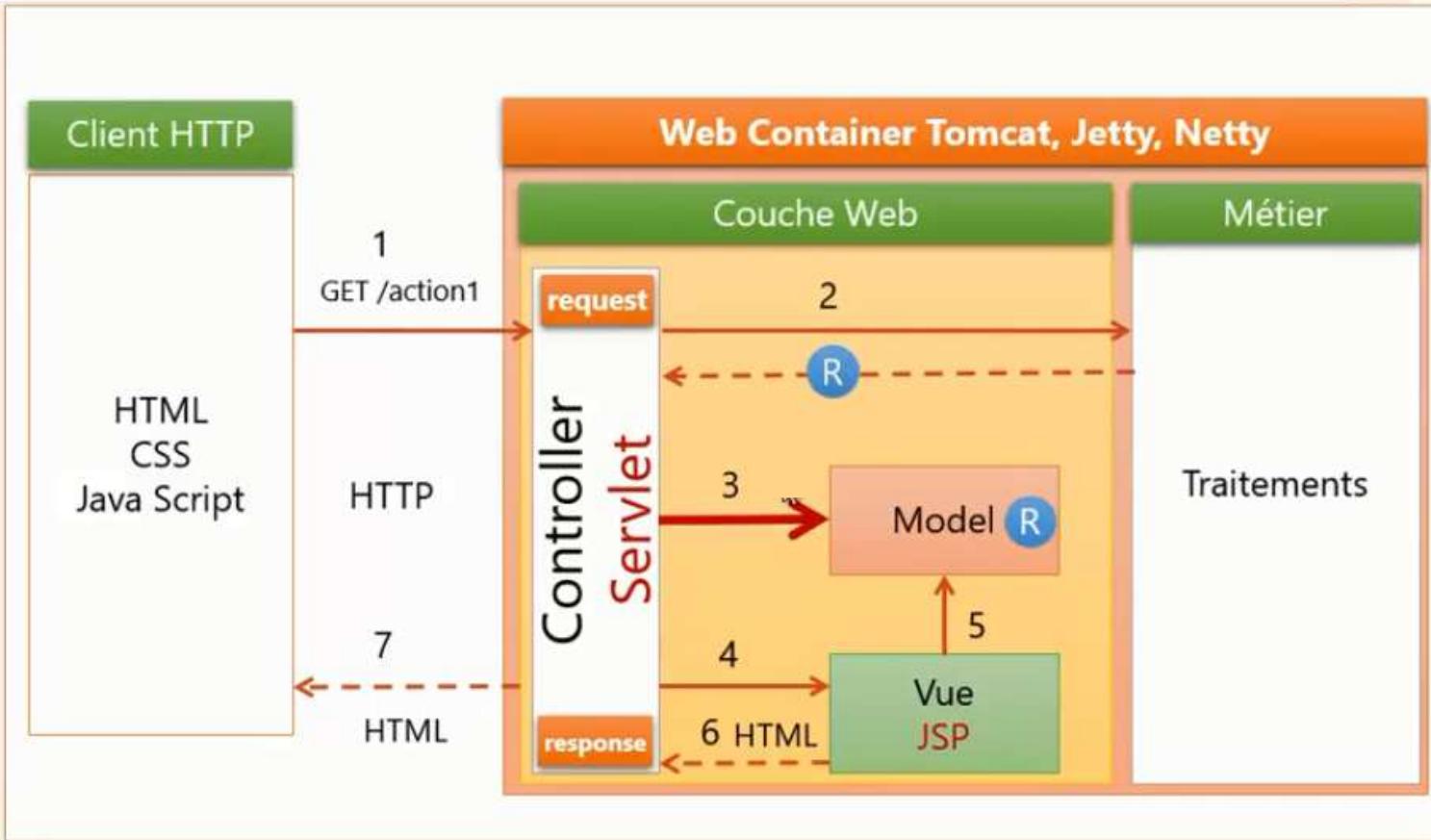
Architecture Web JEE

2 – Le contrôleur fait appel à la couche métier pour effectuer les traitements et récupère les résultats R



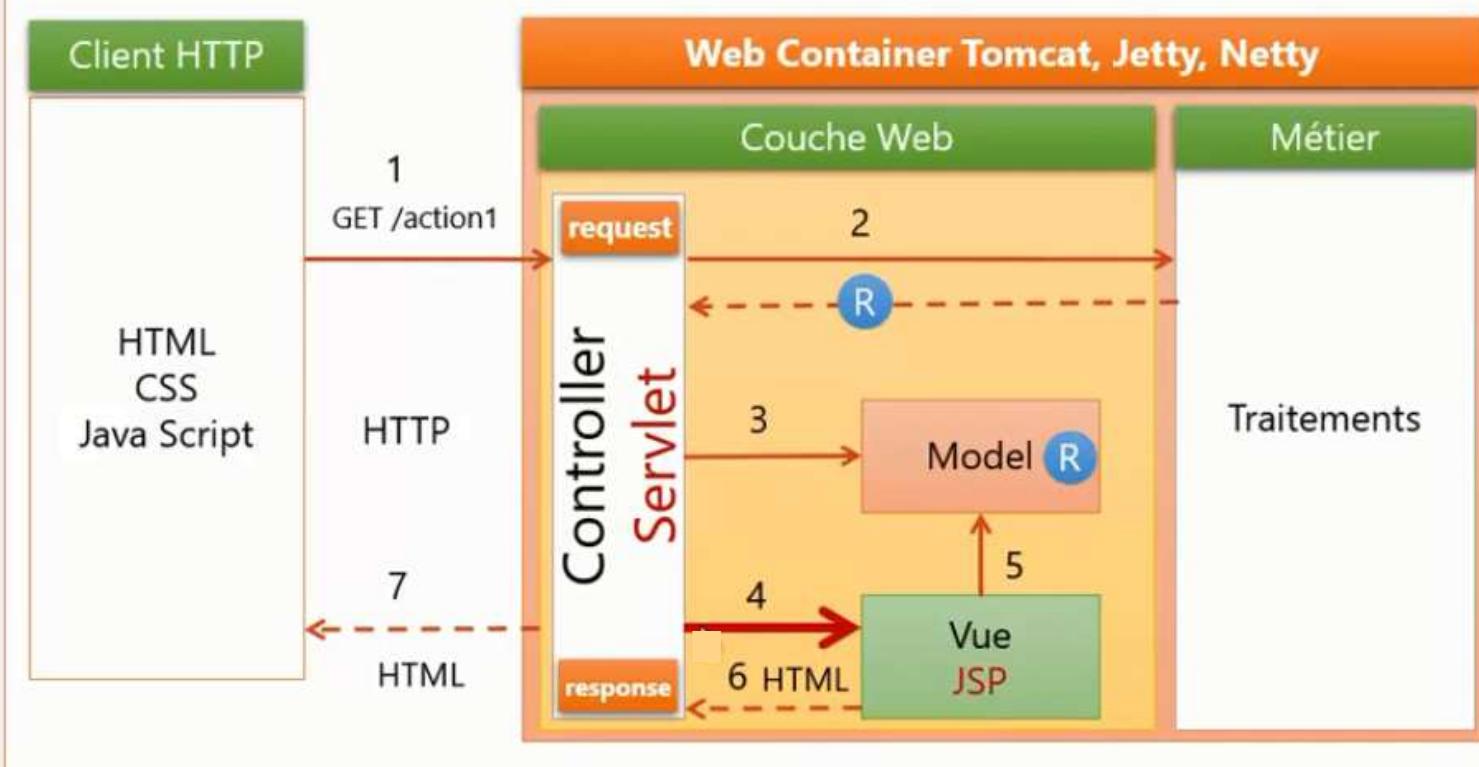
Architecture Web JEE

3 – Le contrôleur Stocke le résultat R dans le modèle M. Le modèle est généralement une objet qui permet de stocker toutes les données qui seront affichées dans la vue. Généralement, le contrôleur stocke le modèle dans l'objet request ou session.



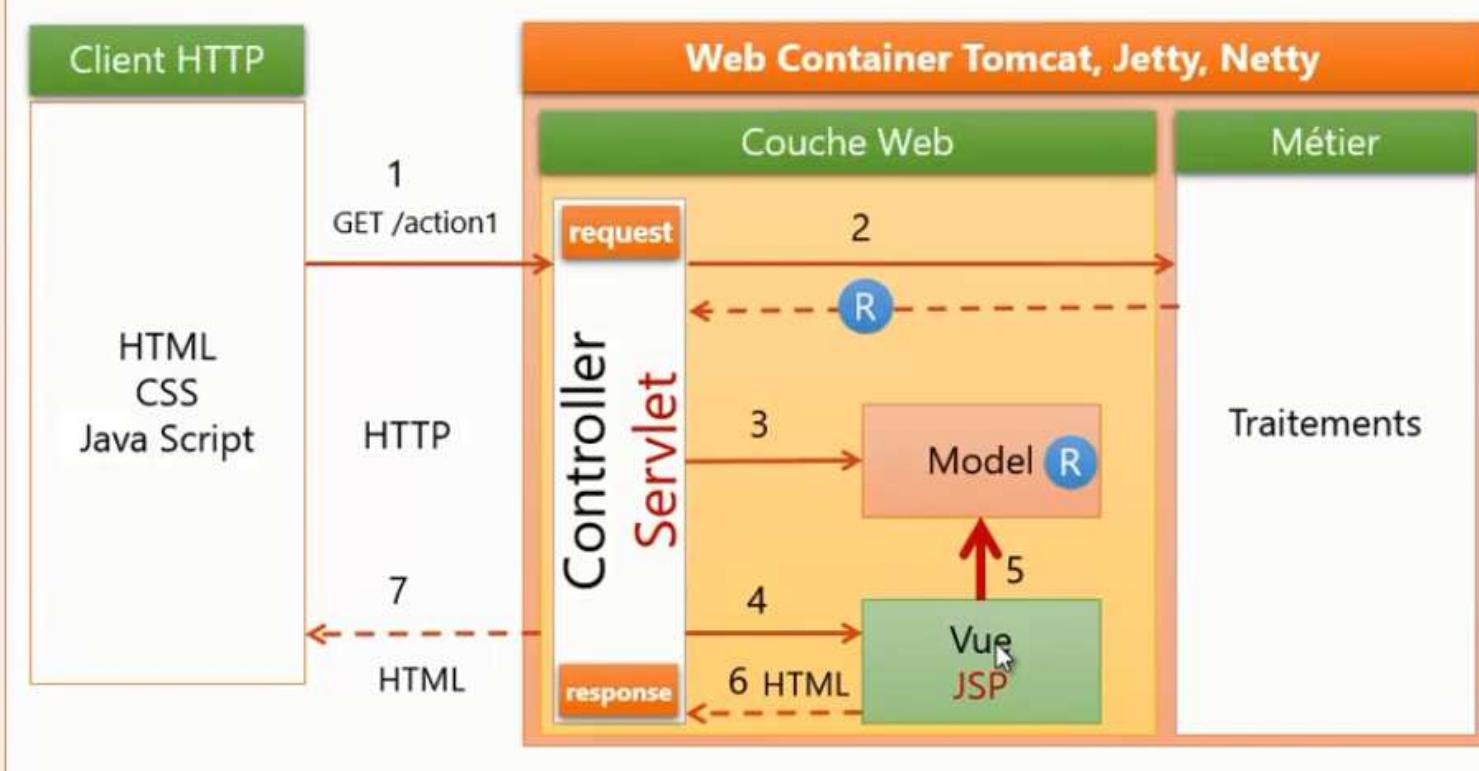
Architecture Web JEE

4 – Le contrôleur fait appel à la vue JSP (Java Server Pages) en lui transmettant les mêmes objets request et response. Cette opération s'appelle Forwarding.



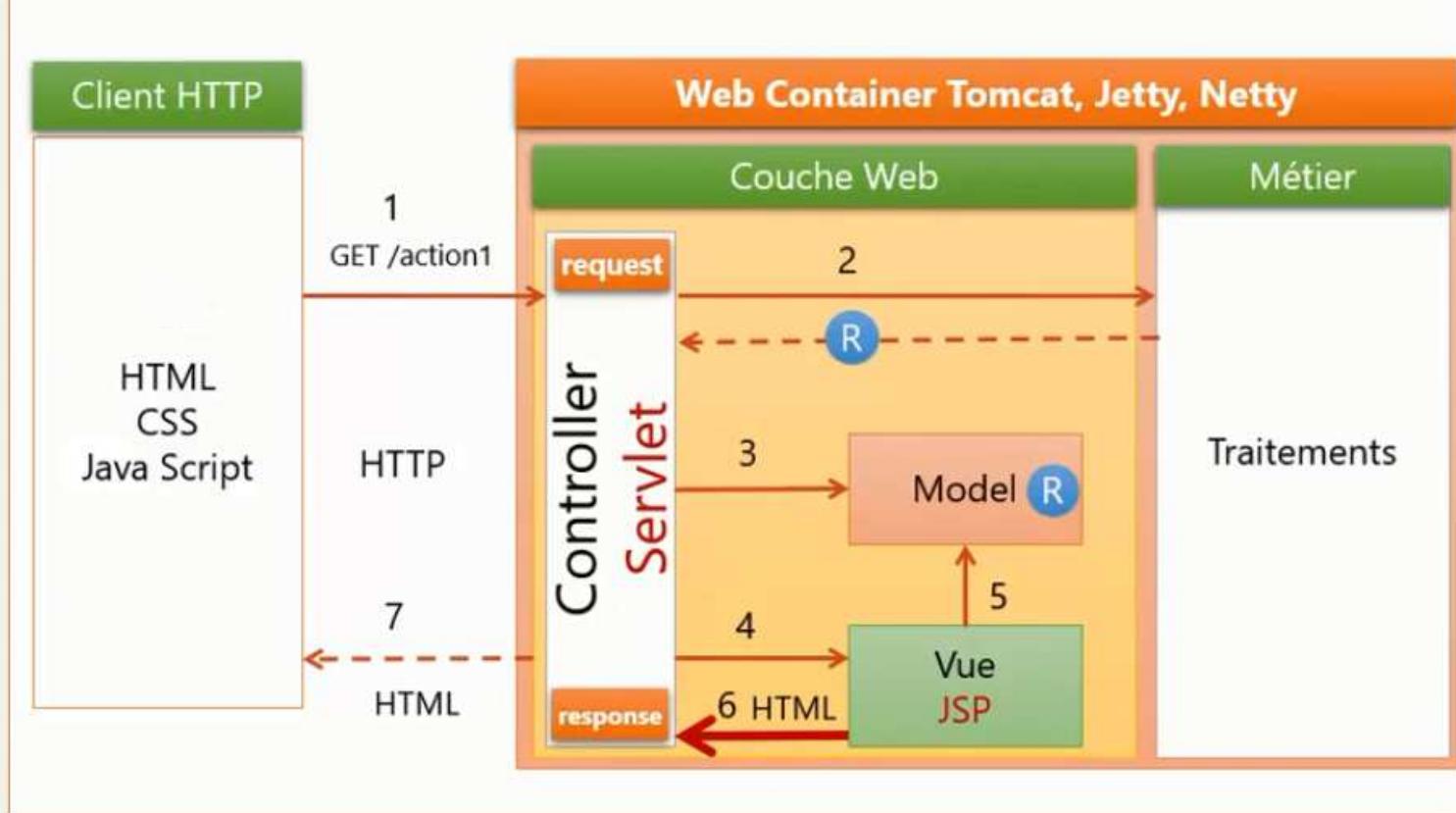
Architecture Web JEE

5 – La vue JSP récupère le résultat à partir du modèle. La vue retrouve le modèle dans l'objet request ou session.



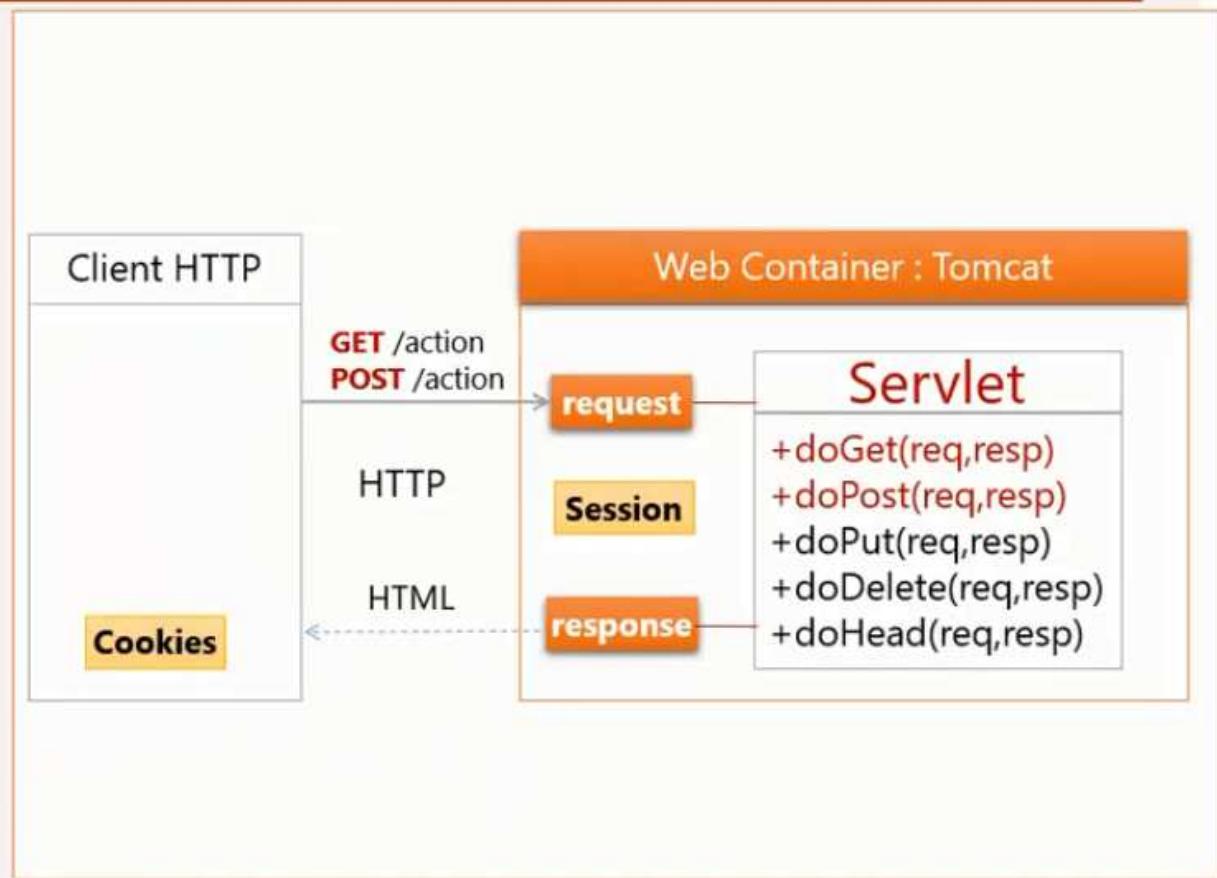
Architecture Web JEE

6 – La vue JSP génère dynamiquement une page HTML qui contient les résultats du modèle en utilisant l'objet response.



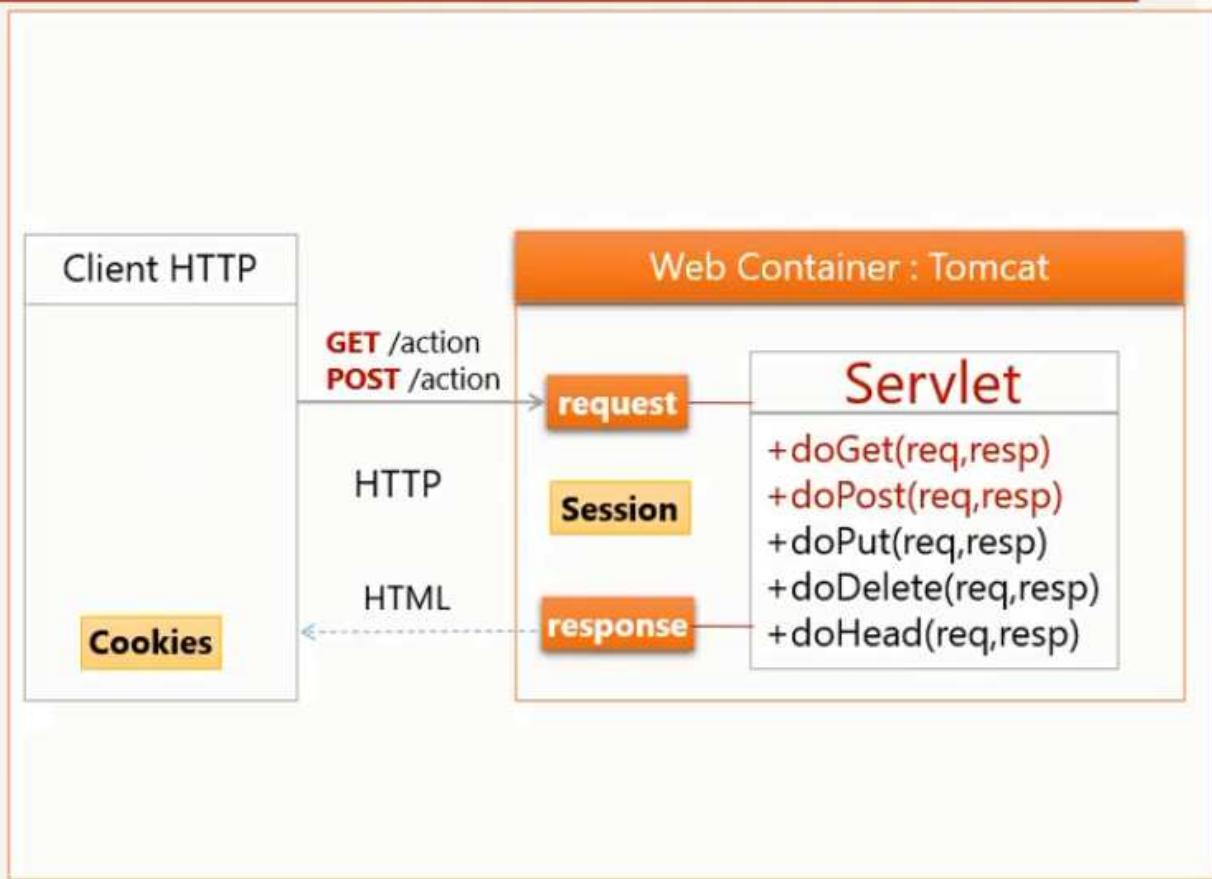
Servlet

- Un Servlet est un composant Web JEE qui permet d'effectuer des traitements du côté du serveur suite à une requête HTTP et envoyer une réponse HTTP.
- Une Servlet est une classe Java qui hérite de HttpServlet et qui redéfinit des méthodes comme doGet, doPost, doPut, doDelete, doHead. Et d'autres méthodes qui définissent son cycle de vie.
- La méthode doX est exécutée si une requête HTTP est envoyée par un client http avec la méthode X

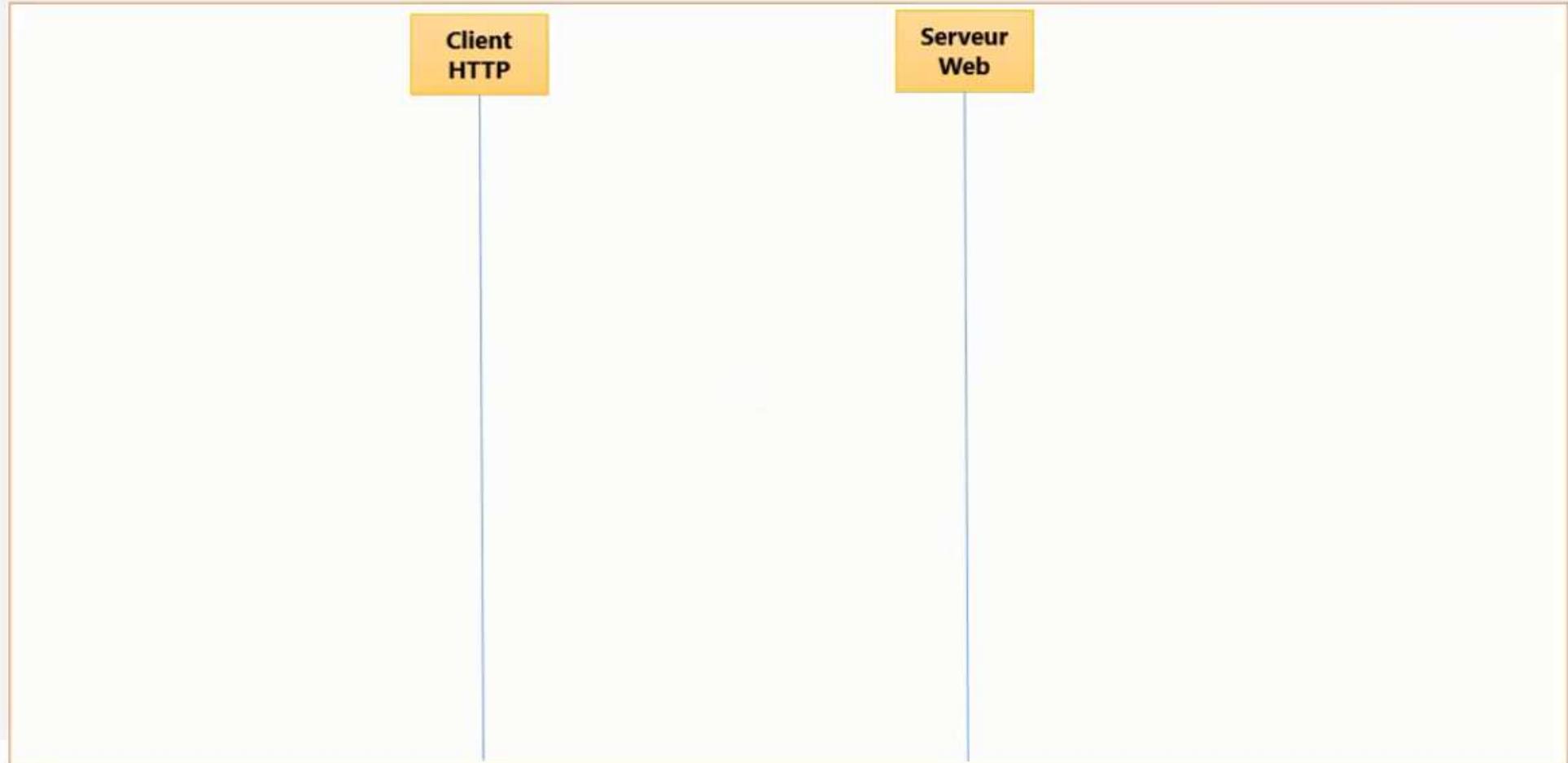


http : Session et Cookies

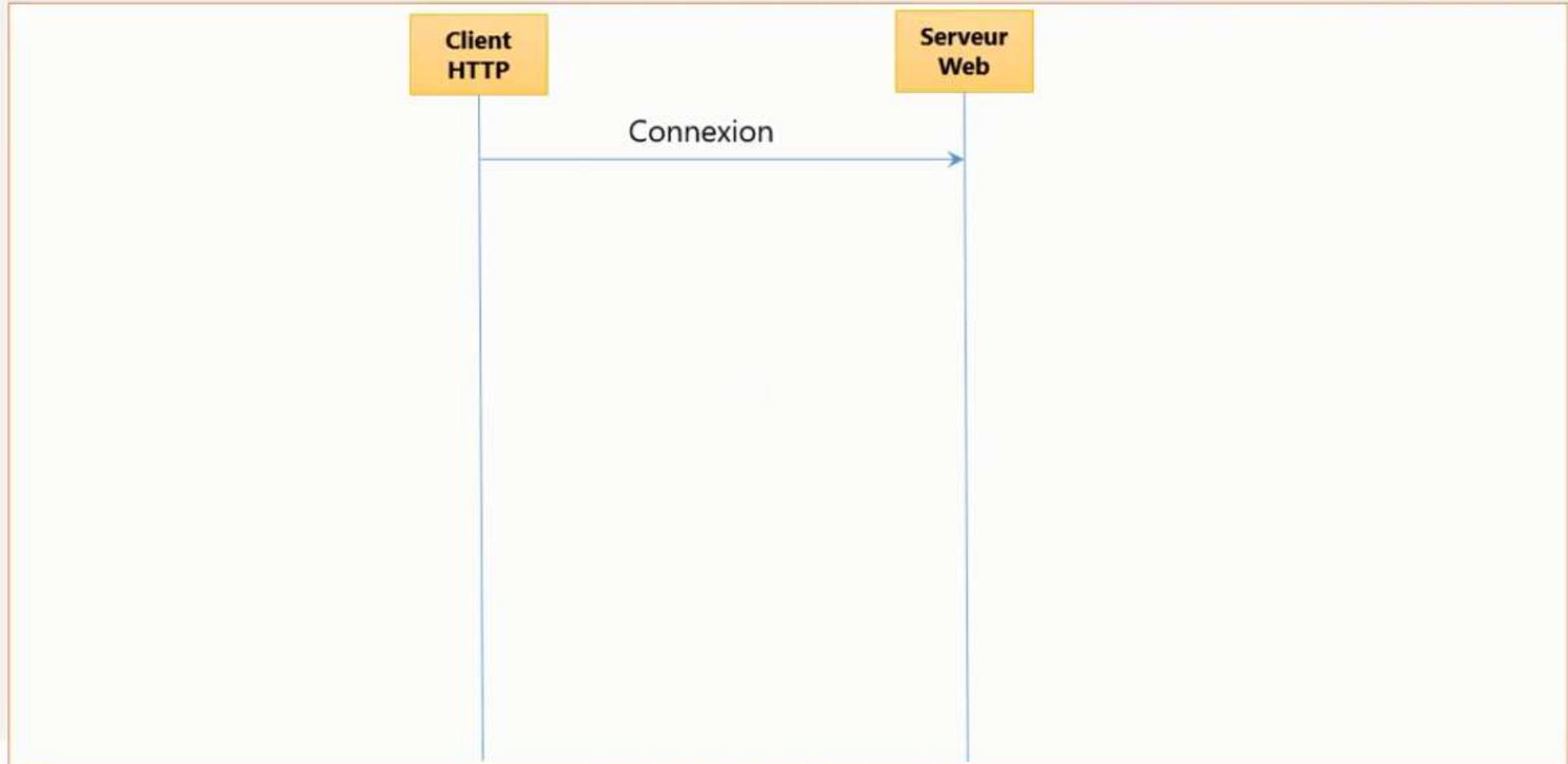
- Pour lire les données de la requête la servlet utilise l'objet request
- Pour envoyée la réponse HTTP, la servlet utilise l'objet response.
- Pour chaque nouveau client HTTP, le serveur crée un objet Session qui permet de stocker les données relatives au client dans la mémoire du serveur.
- Une session possède un timeout (20 min) au bout duquel, si le client d'envoie pas de requête HTTP, la session est détruite.
- dans une réponse HTTP, le serveur peut demander au client d'enregistrer des données relatives au client, dans des fichiers de la machine du client. Ces fichiers s'appellent les cookies. Les cookies ont également une durée de vie. Quand cette durée expire, les cookies sont détruit par le navigateur web



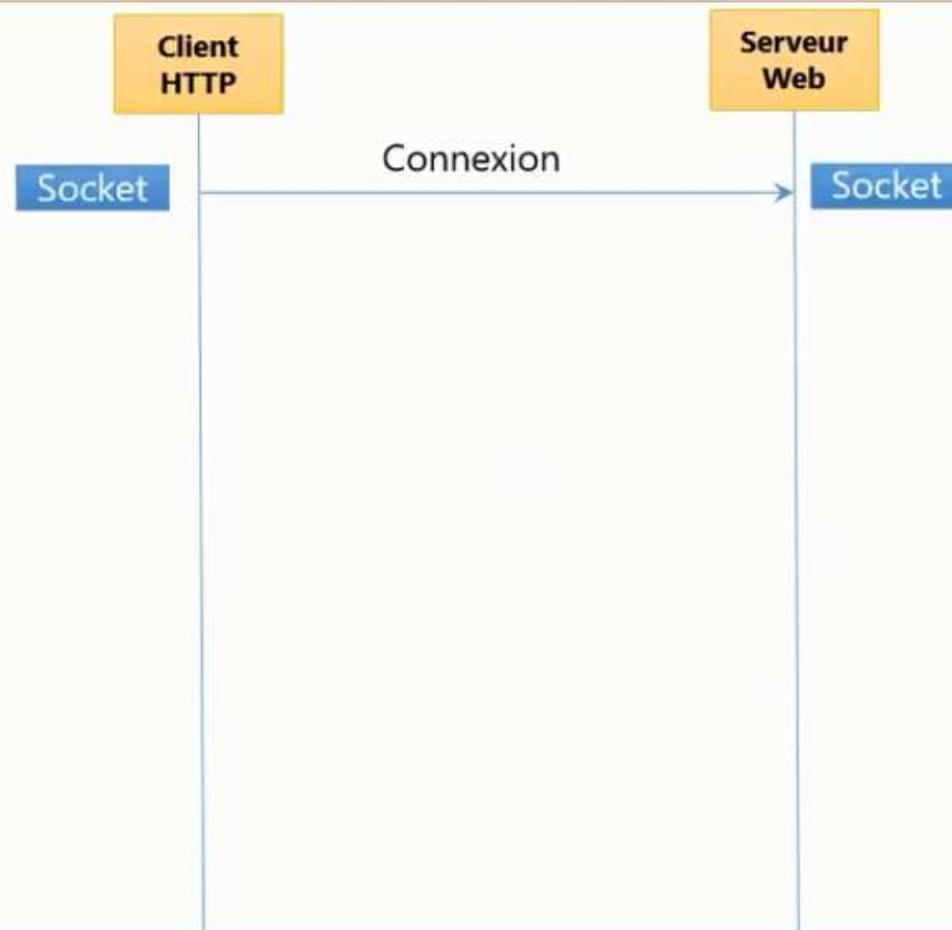
Session et Cookies



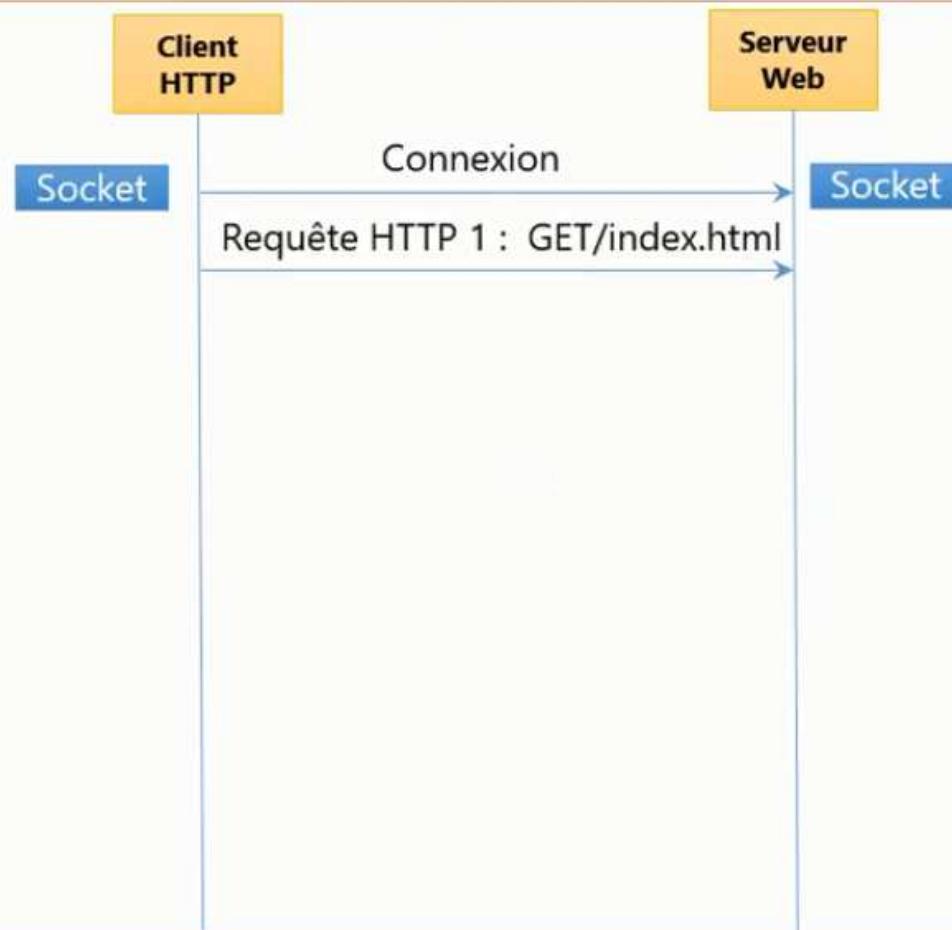
Session et Cookies



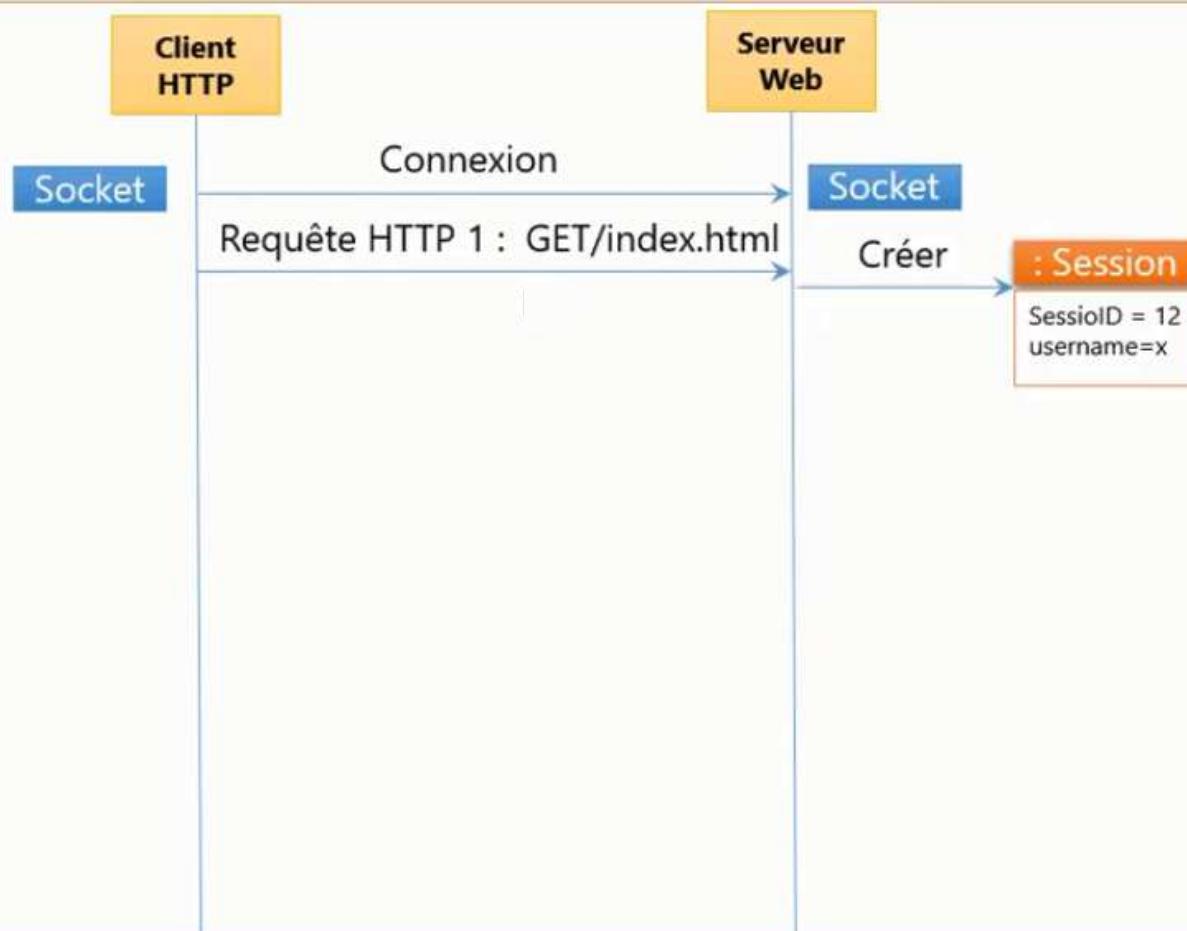
Session et Cookies



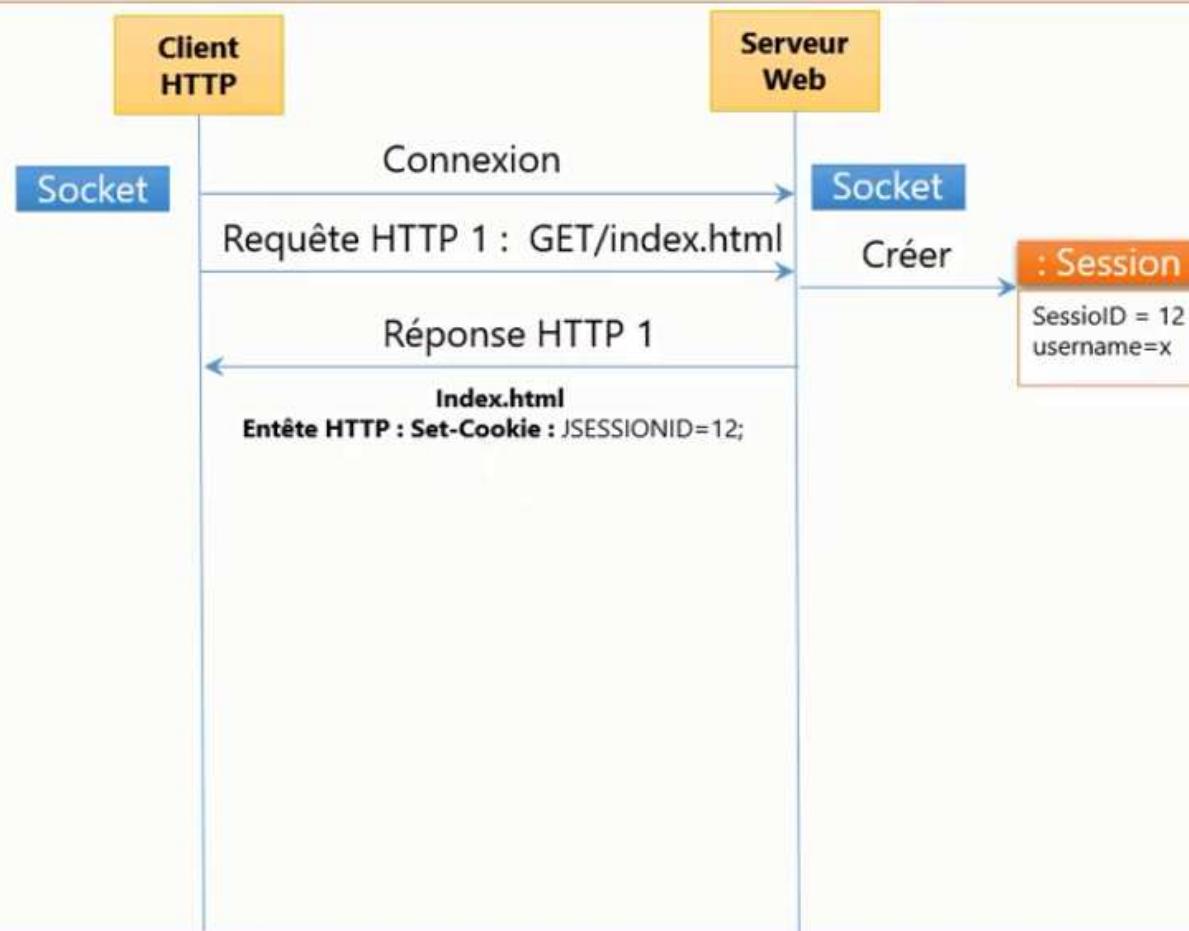
Session et Cookies



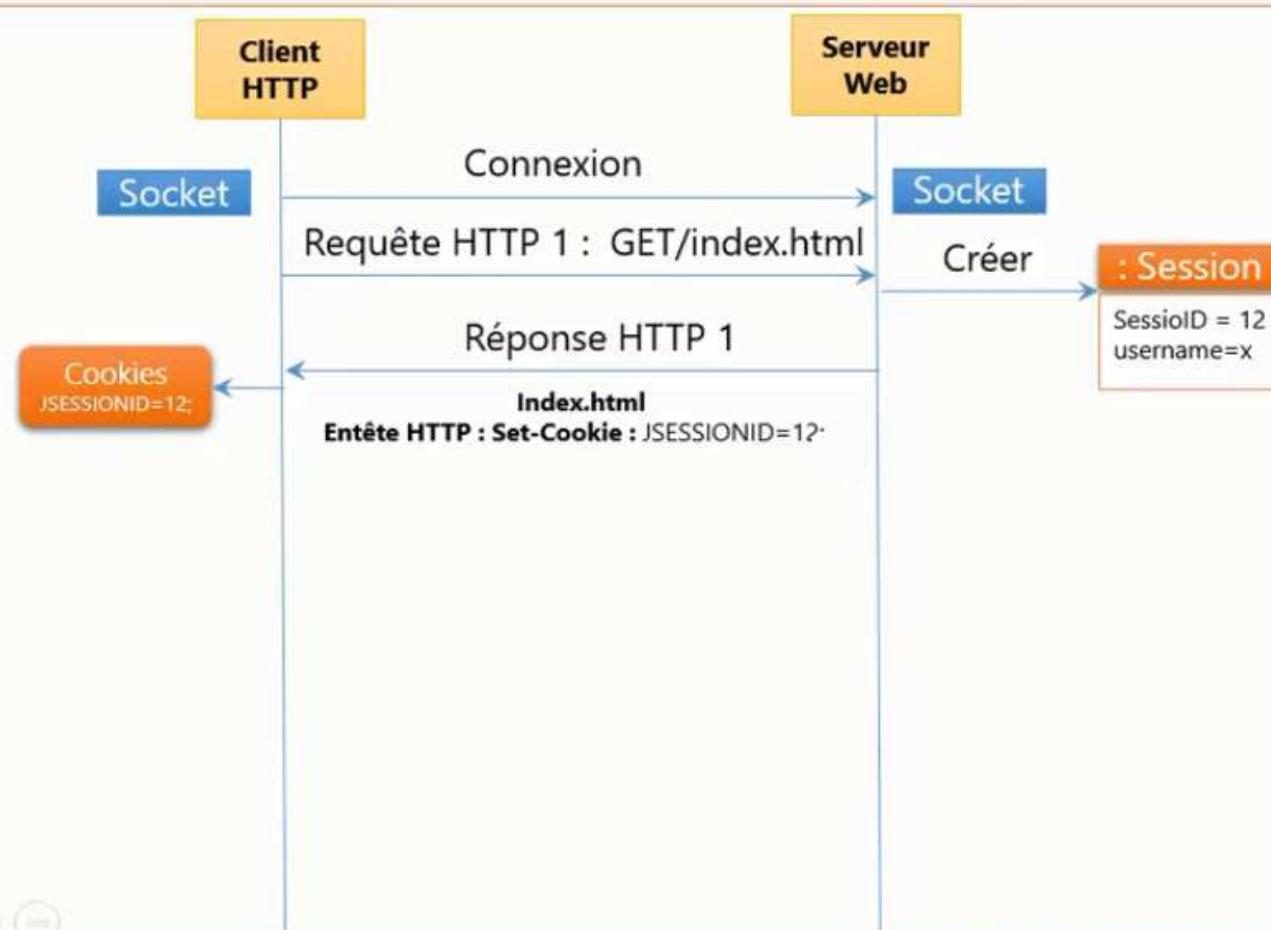
Session et Cookies



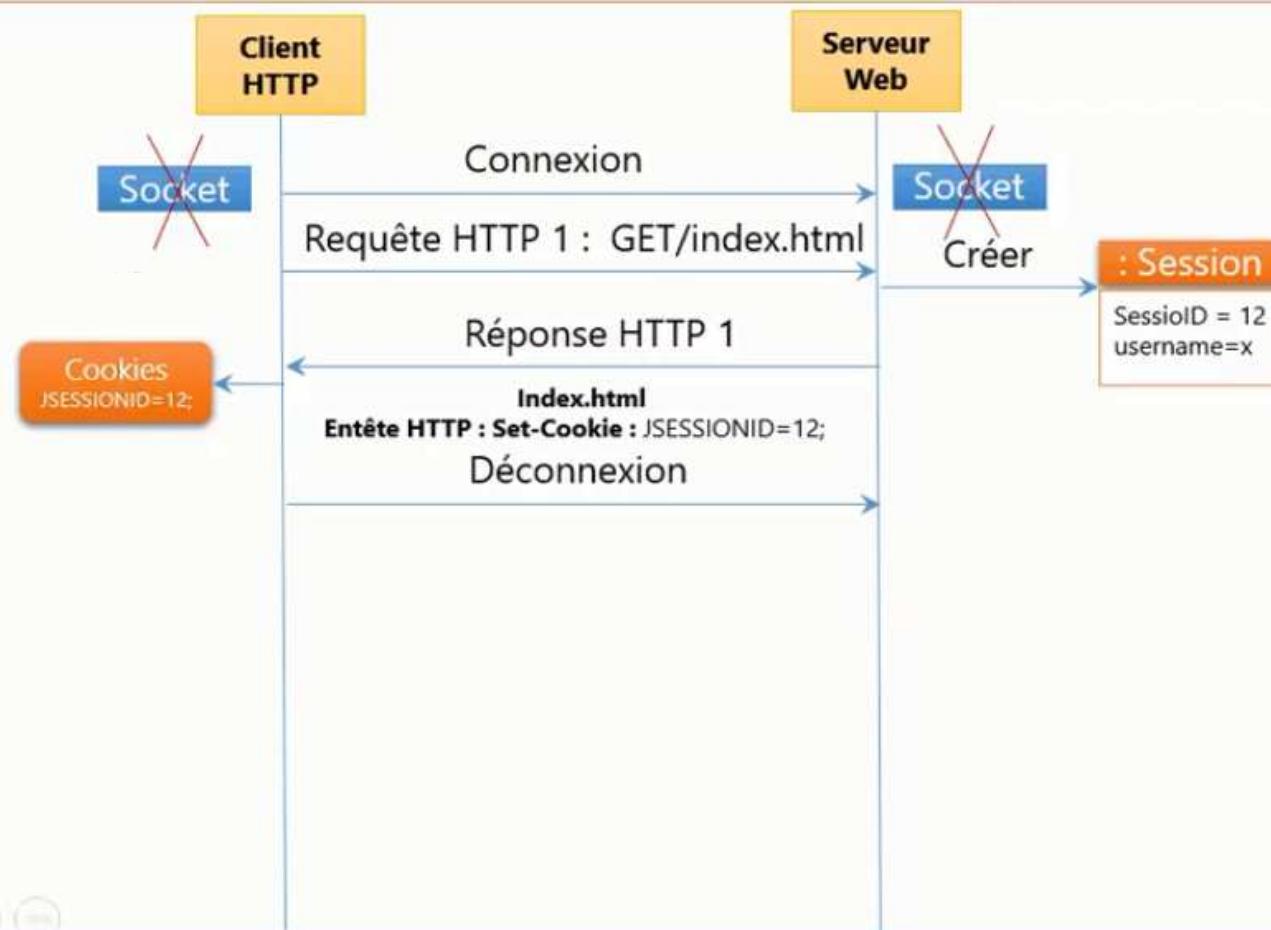
Session et Cookies



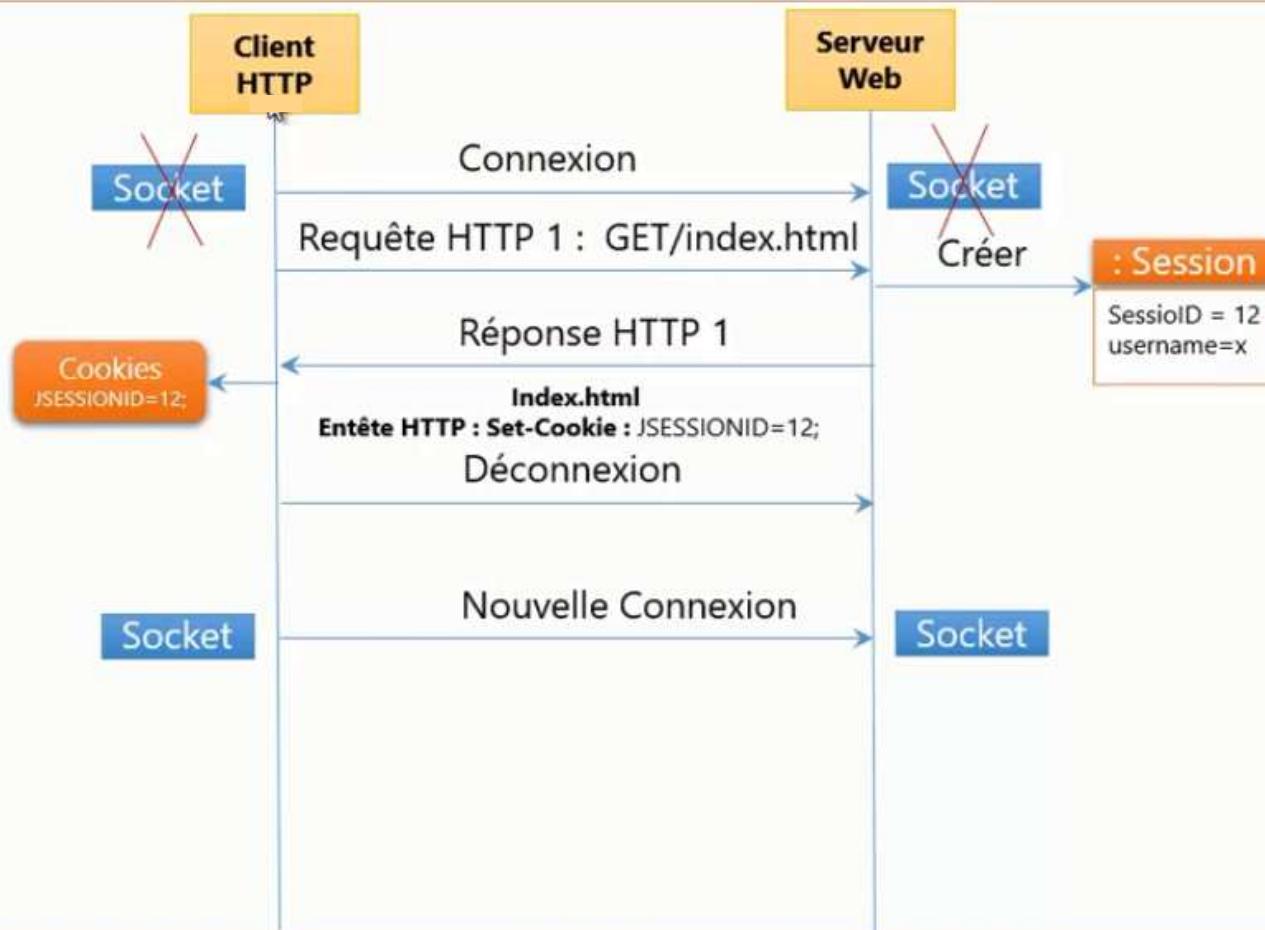
Session et Cookies



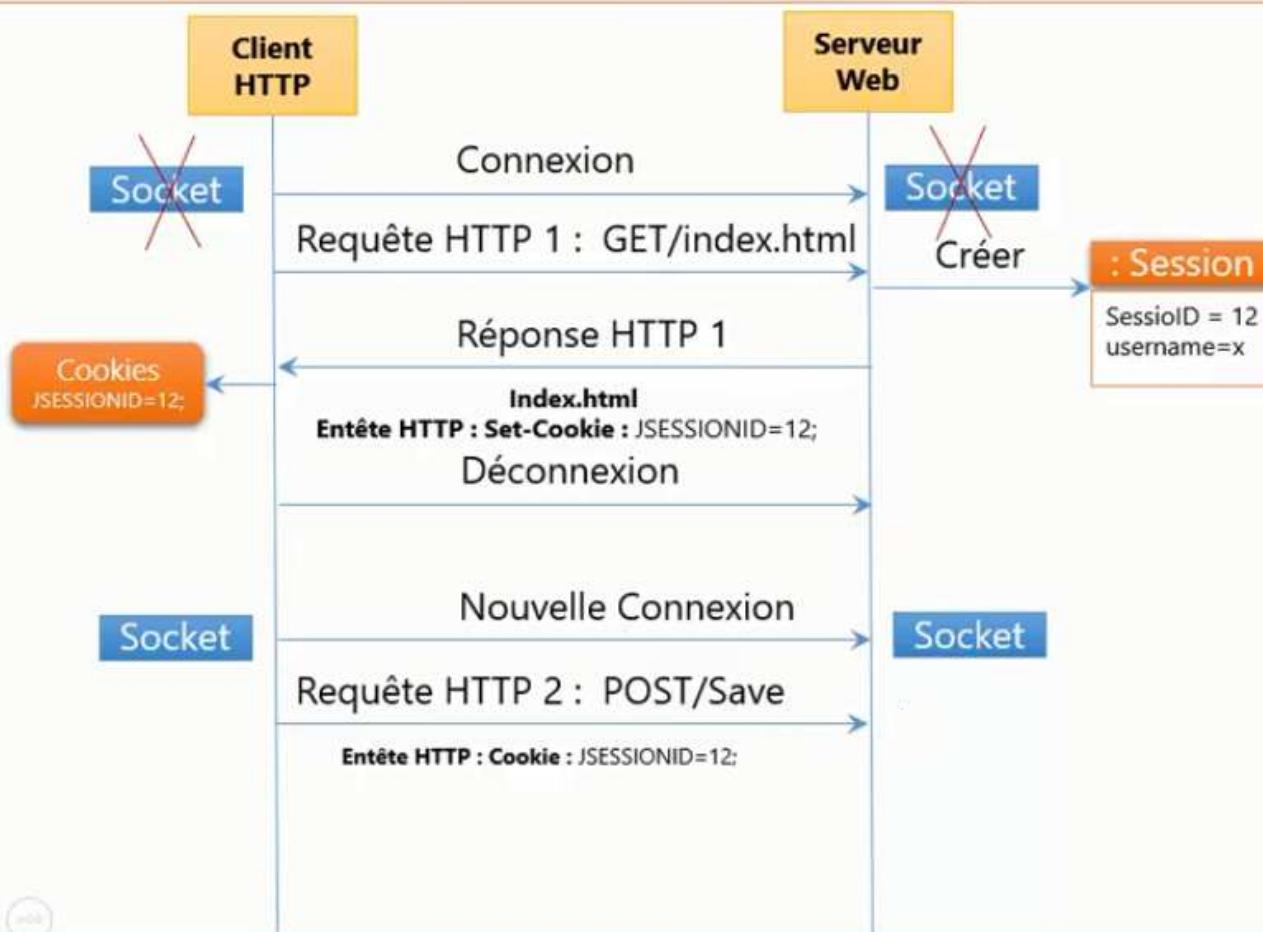
Session et Cookies



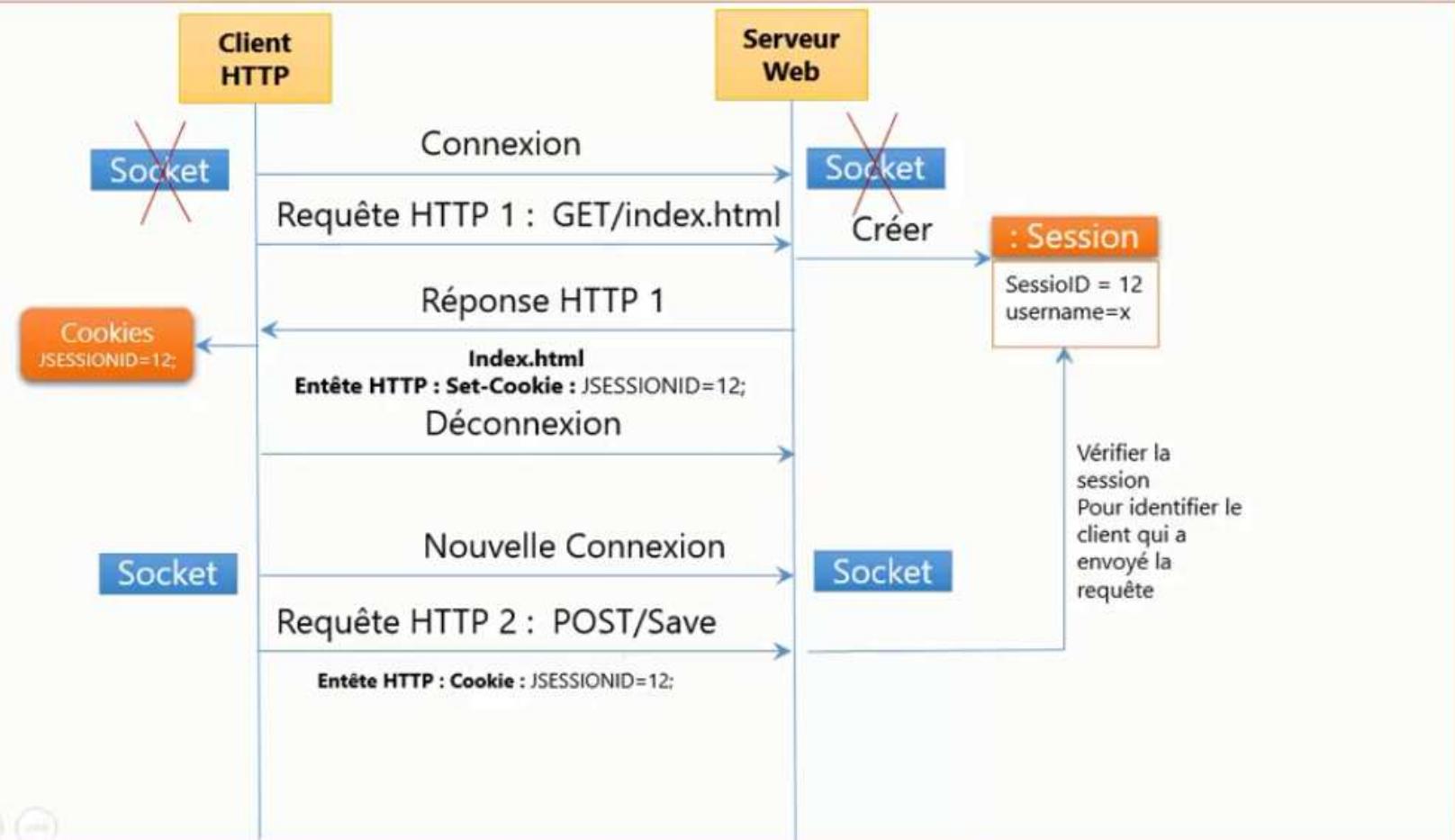
Session et Cookies



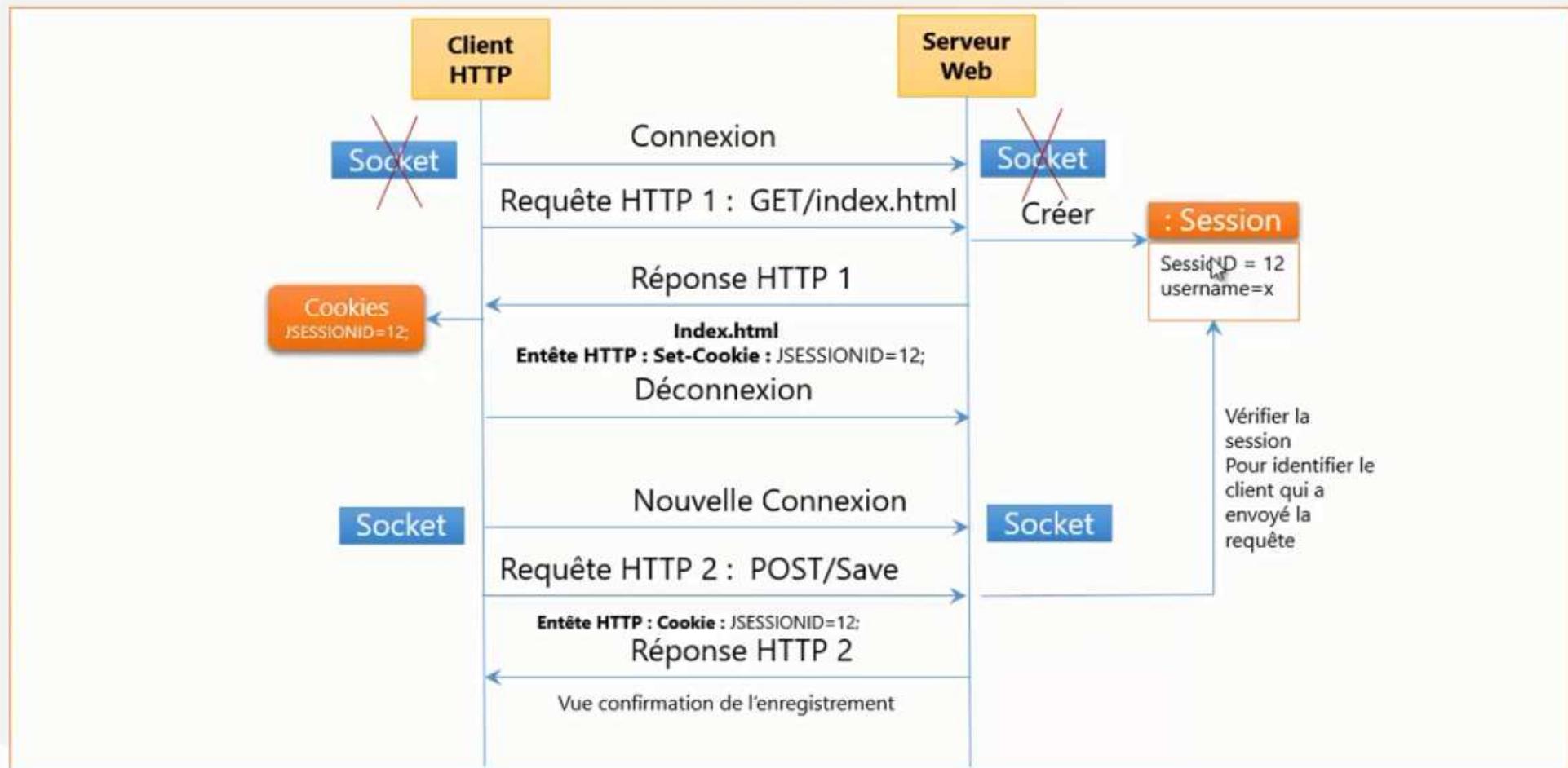
Session et Cookies



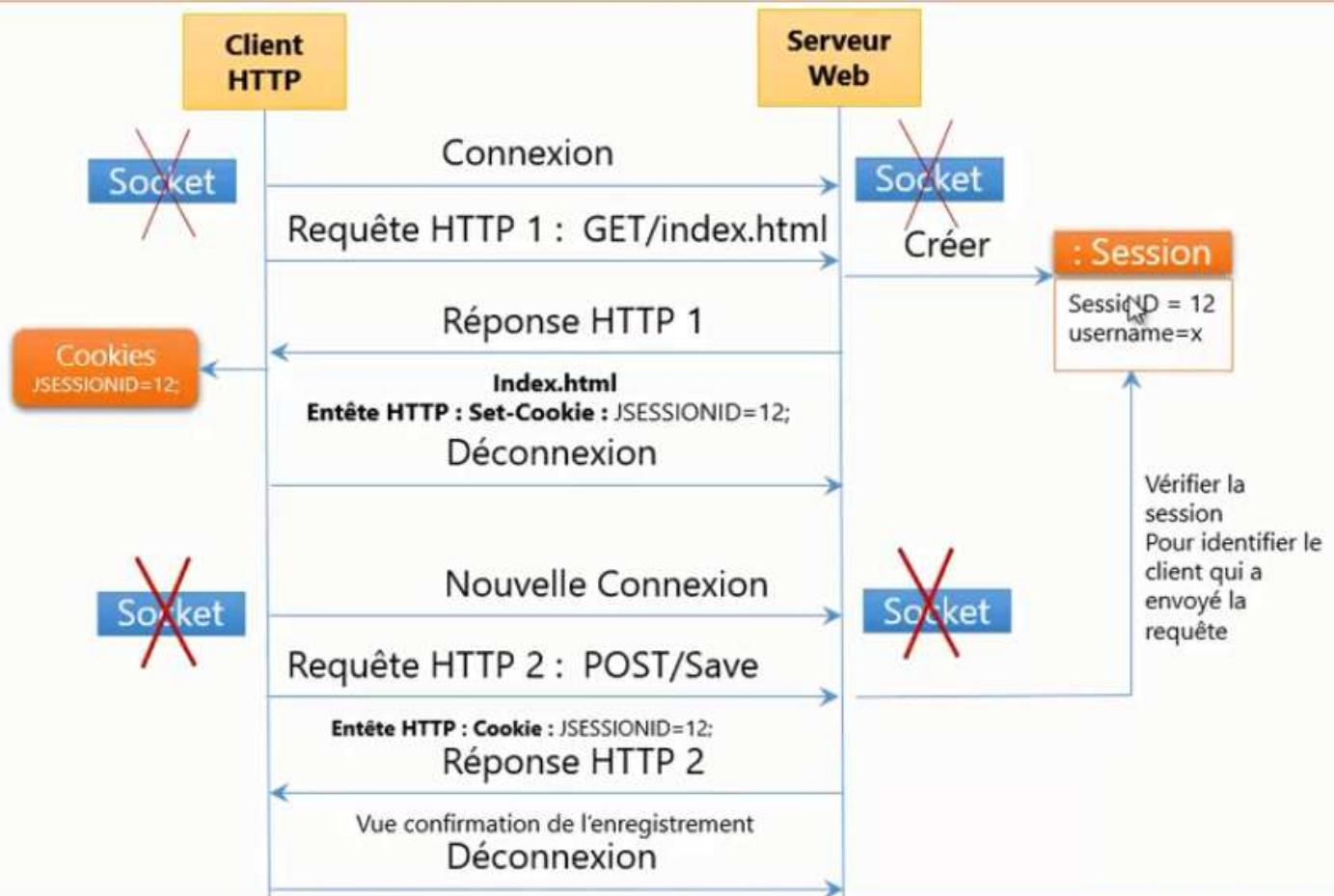
Session et Cookies



Session et Cookies



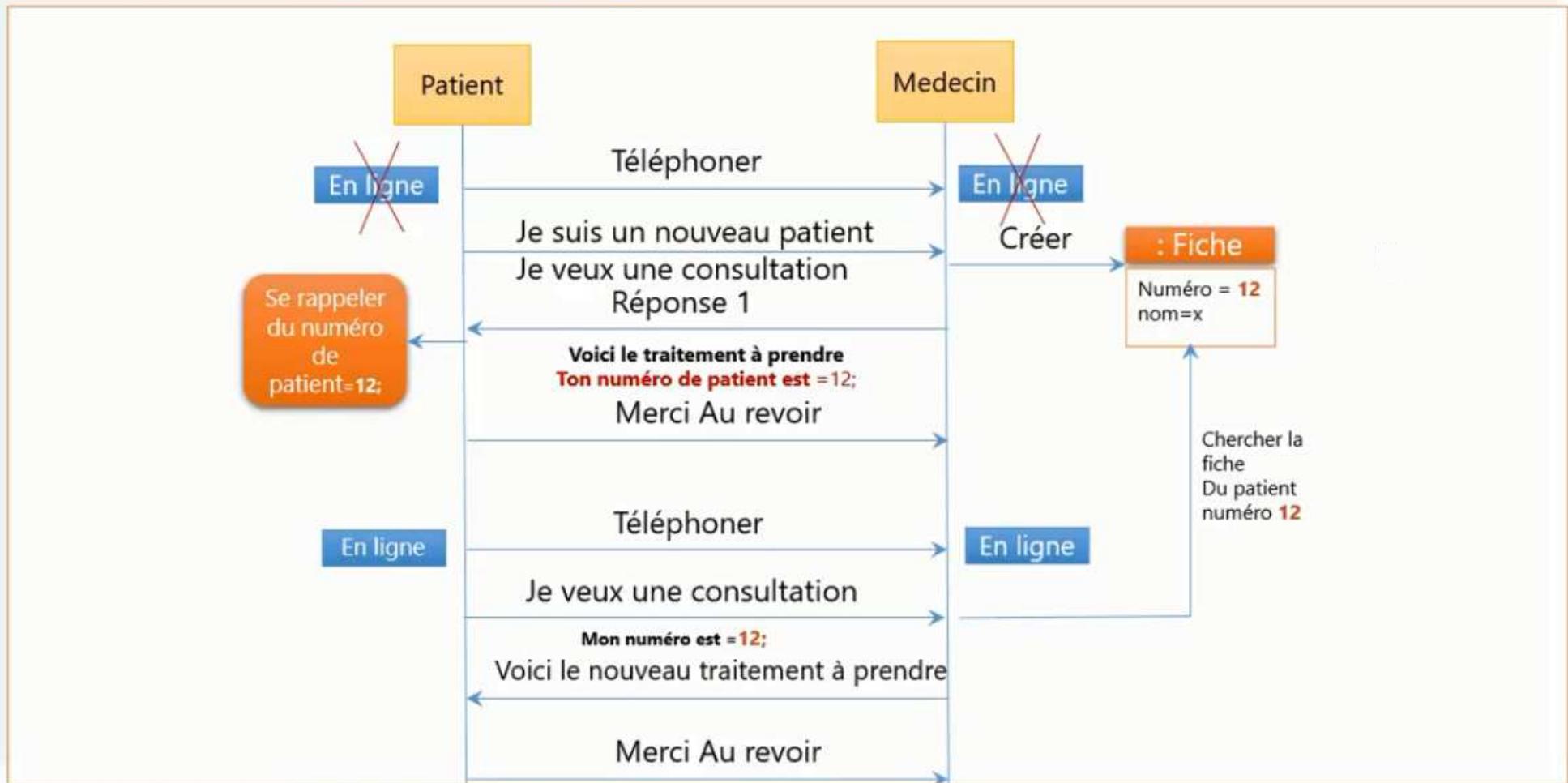
Session et Cookies



Utilisation des sessions et des cookies

- Généralement quand un client HTTP envoie sa première requête, le serveur web crée une session pour ce client.
- Une session est un objet stocké dans la mémoire du serveur qui peut servir pour stocker des informations relatives au client.
- Le serveur attribut un SessionID unique à chaque session.
- Ce SessionID est ensuite envoyé dans la réponse http en sousforme d'un cookie en utilisant l'entête de la réponse HTTP :
- **Set-Cookie : JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11;**
- Ce qui signifie que le serveur demande au client d'enregistrer ce SESSIONID dans un fichier stocké dans la machine du client appelé COOKIE.
- Une fois que le client reçoive la réponse HTTP, la connexion est fermée.
- A chaque fois que le client envoie une requête HTTP vers le serveur, il envoie toujours les données des cookies dont le SESSIONID.
- Les cookies sont envoyés dans la requête HTTP en utilisant une entête COOKIE:
- **Cookie: JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11**
- Grâce à cette information, le serveur peut savoir de quel client s'agit-il même s'il s'agit d'une nouvelle connexion.

Session et Cookies > Patient et médecin



Structure d'un Servlet

```
package web; import java.io.IOException; import javax.servlet.*; import javax.servlet.http.*;
public class ControleurServlet extends HttpServlet{
    @Override
    public void init() throws ServletException {
        // Initialisation
        // Exécutée juste après instantiation de la servlet par le serveur Tomcat
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Traitement effectué si une requête Http est envoyée avec GET
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        // Traitement effectué si une requête Http est envoyée avec POST
    }
    @Override
    public void destroy() {
        // Exécutée juste avant la destruction de la servlet.
        // Au moment de l'arrêt de l'application
    }
}
```

HttpServlet

MyServlet

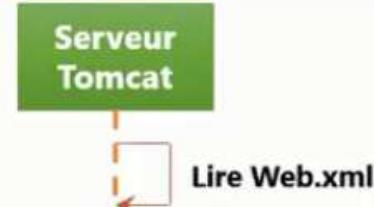
+doGet(req,resp)
+doPost(req,resp)
+init()
+destroy()

Servlet Life Cycle

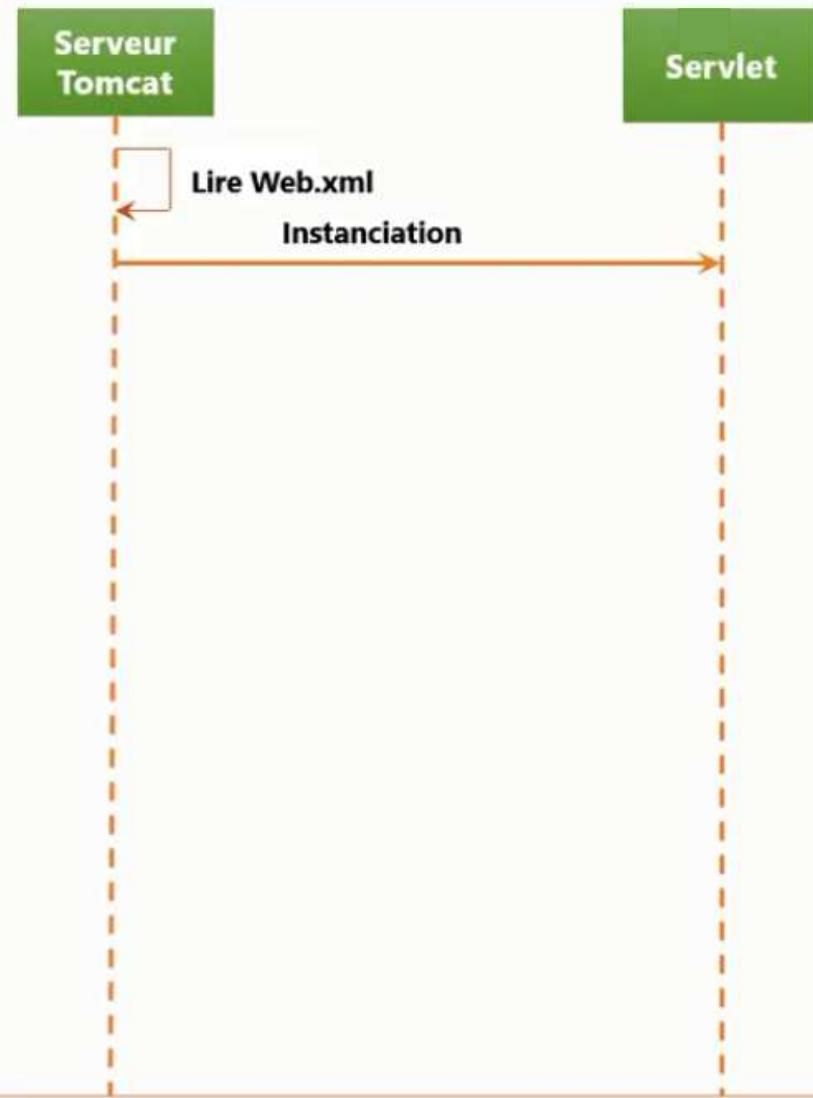
Serveur
Tomcat



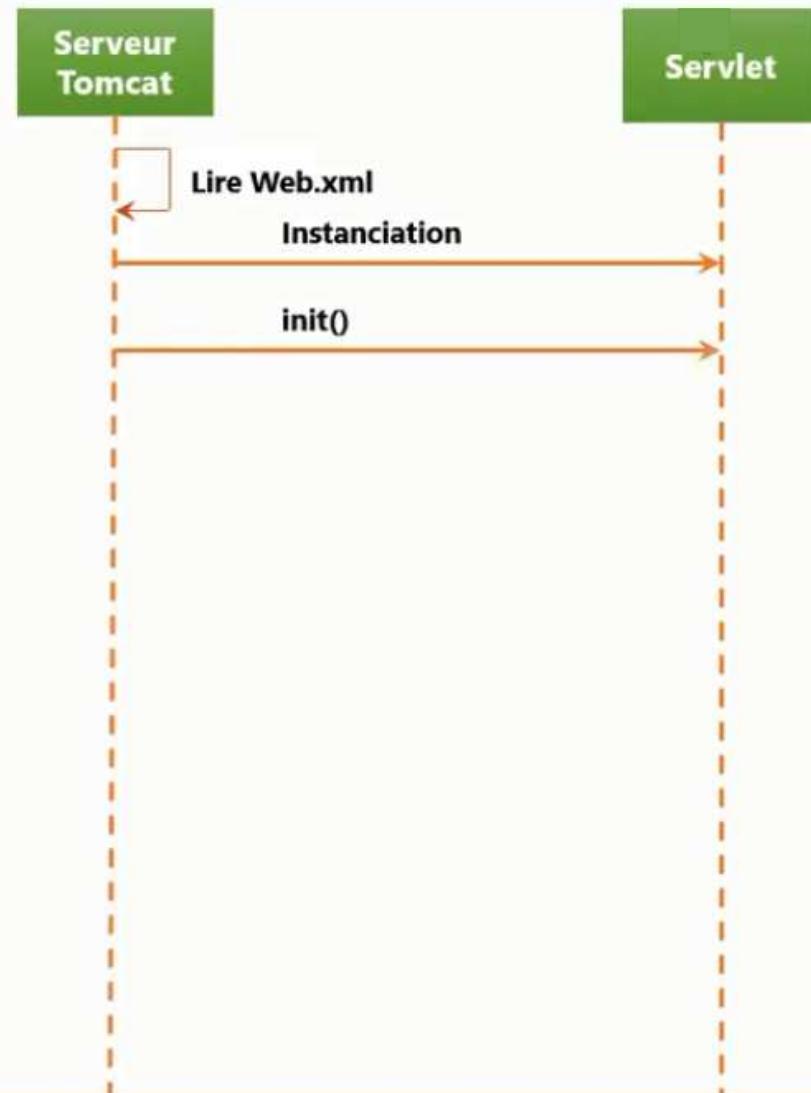
Servlet Life Cycle



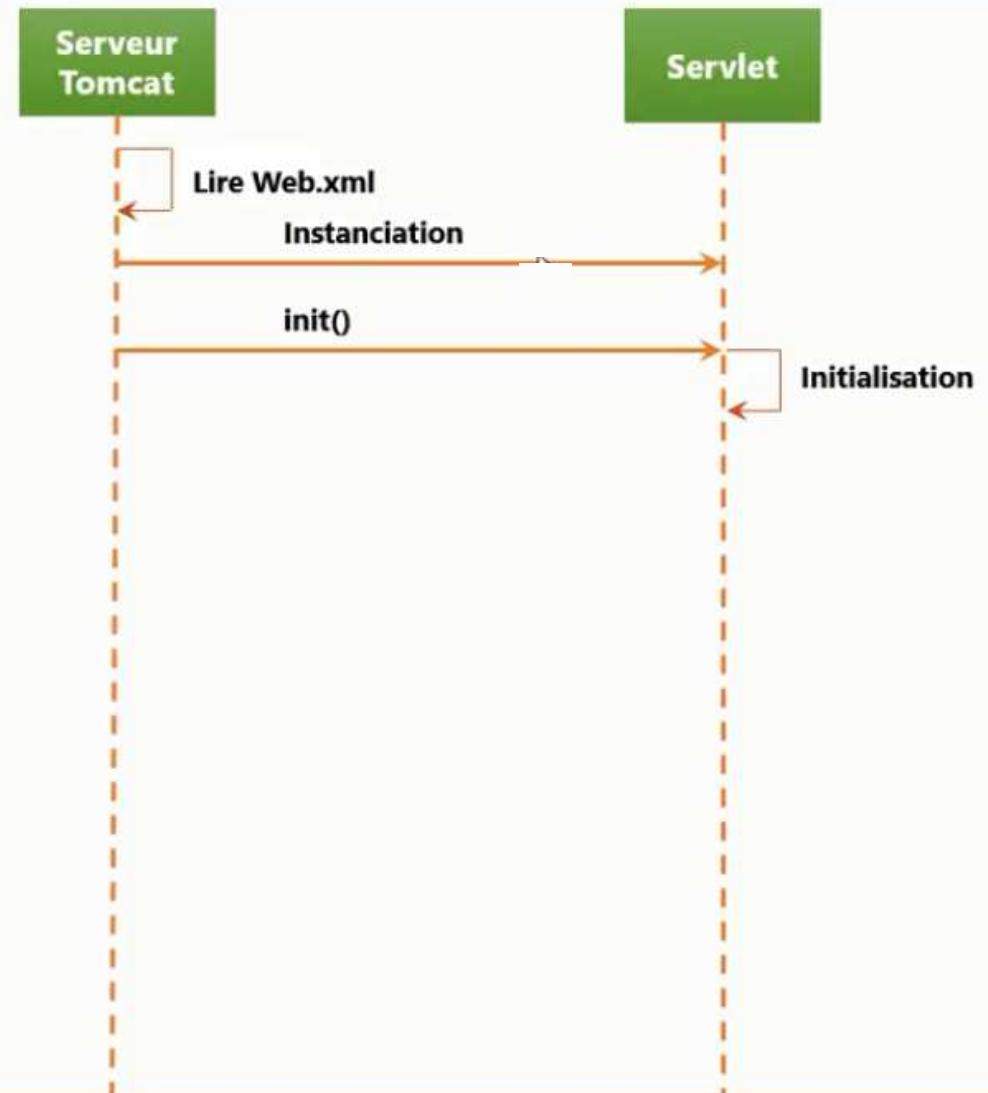
Servlet Life Cycle



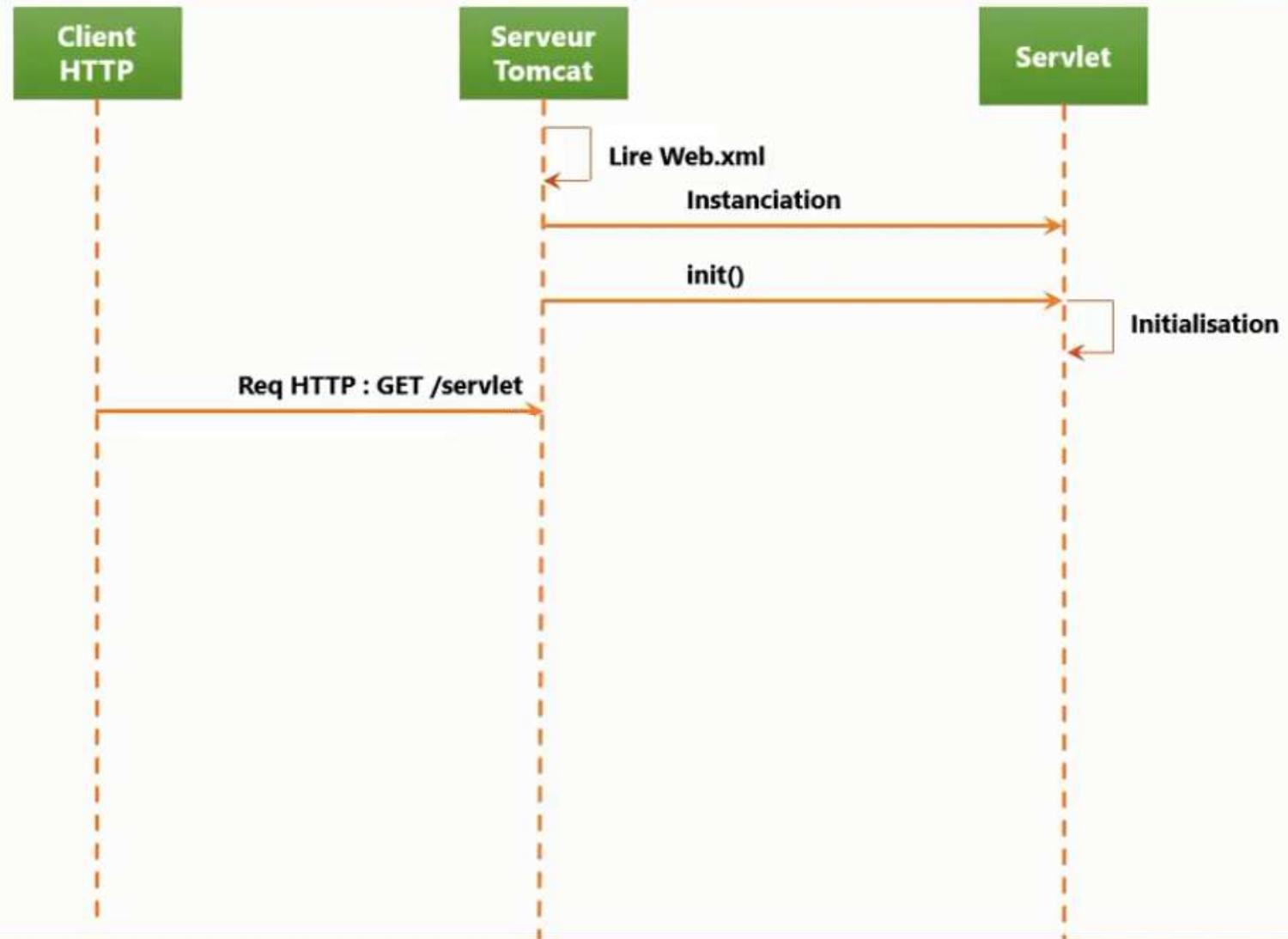
Servlet Life Cycle



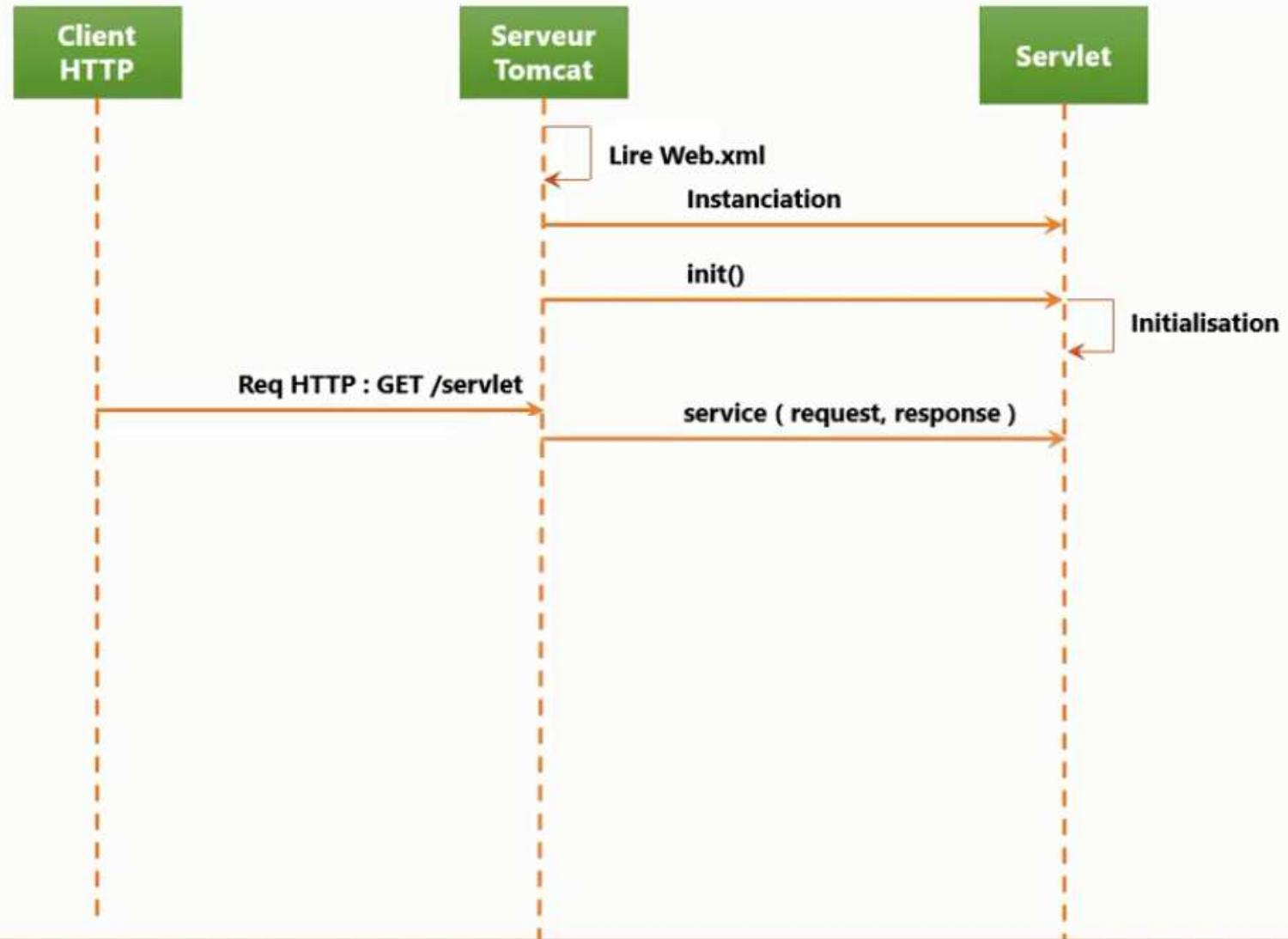
Servlet Life Cycle



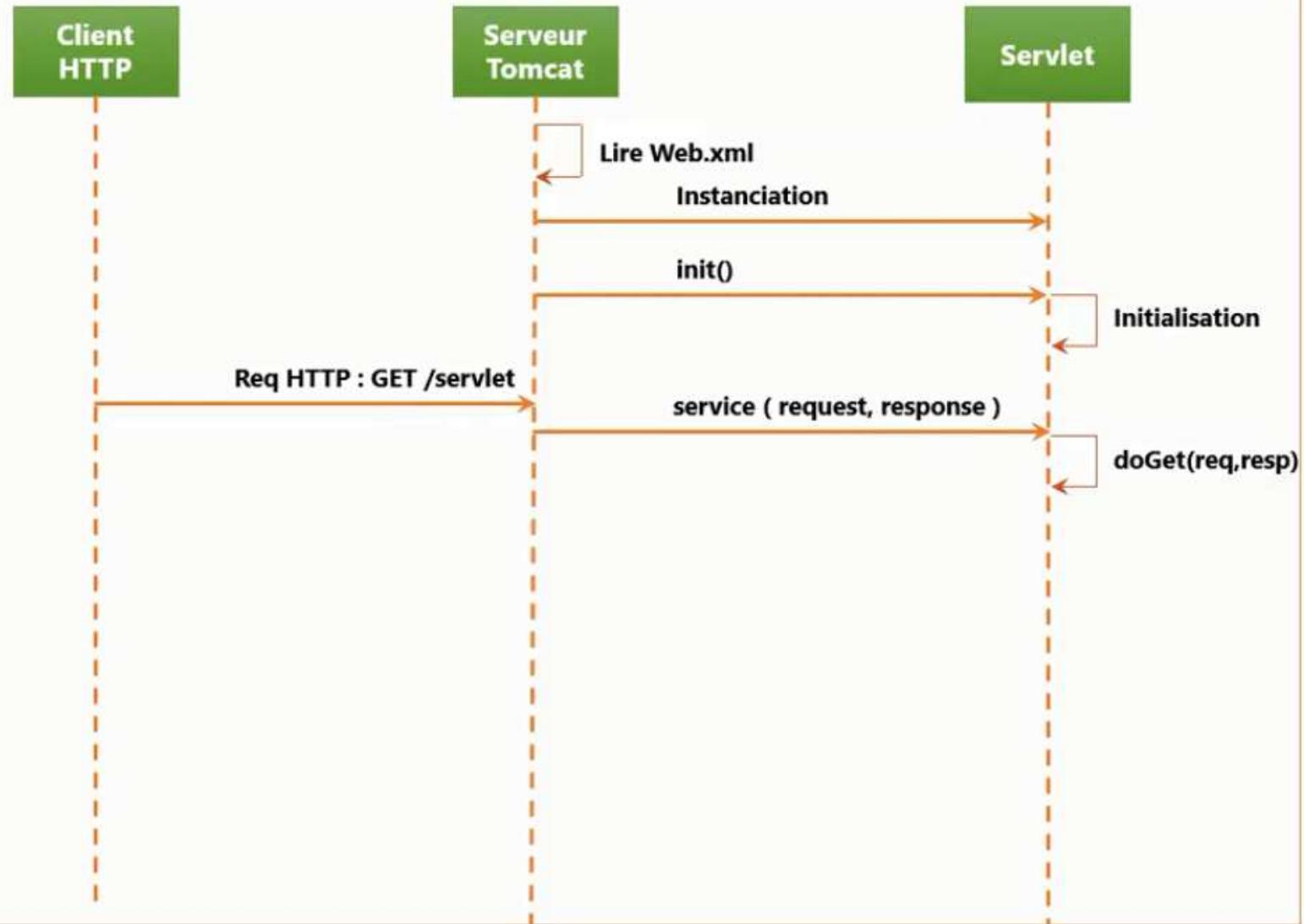
Servlet Life Cycle



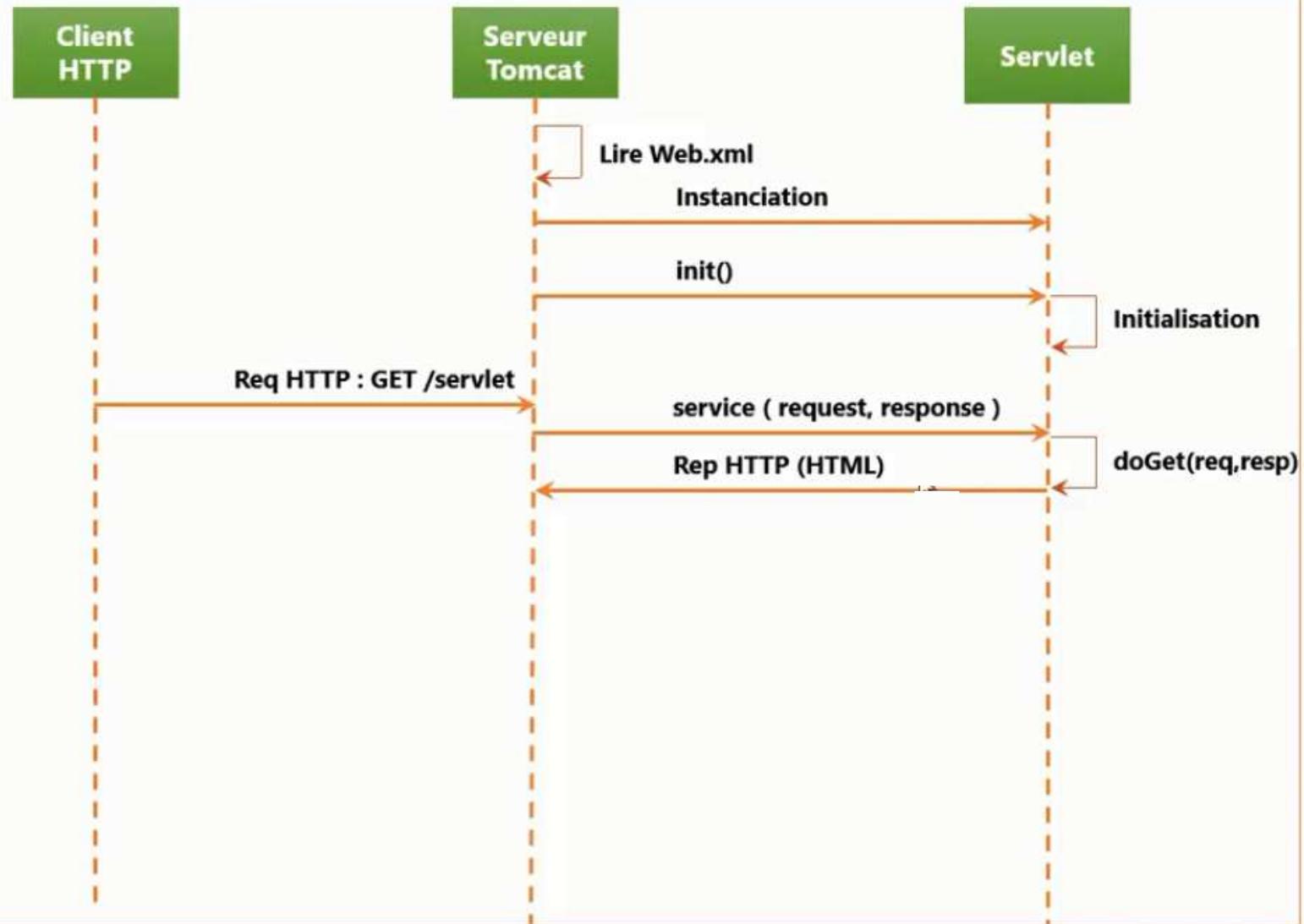
Servlet Life Cycle



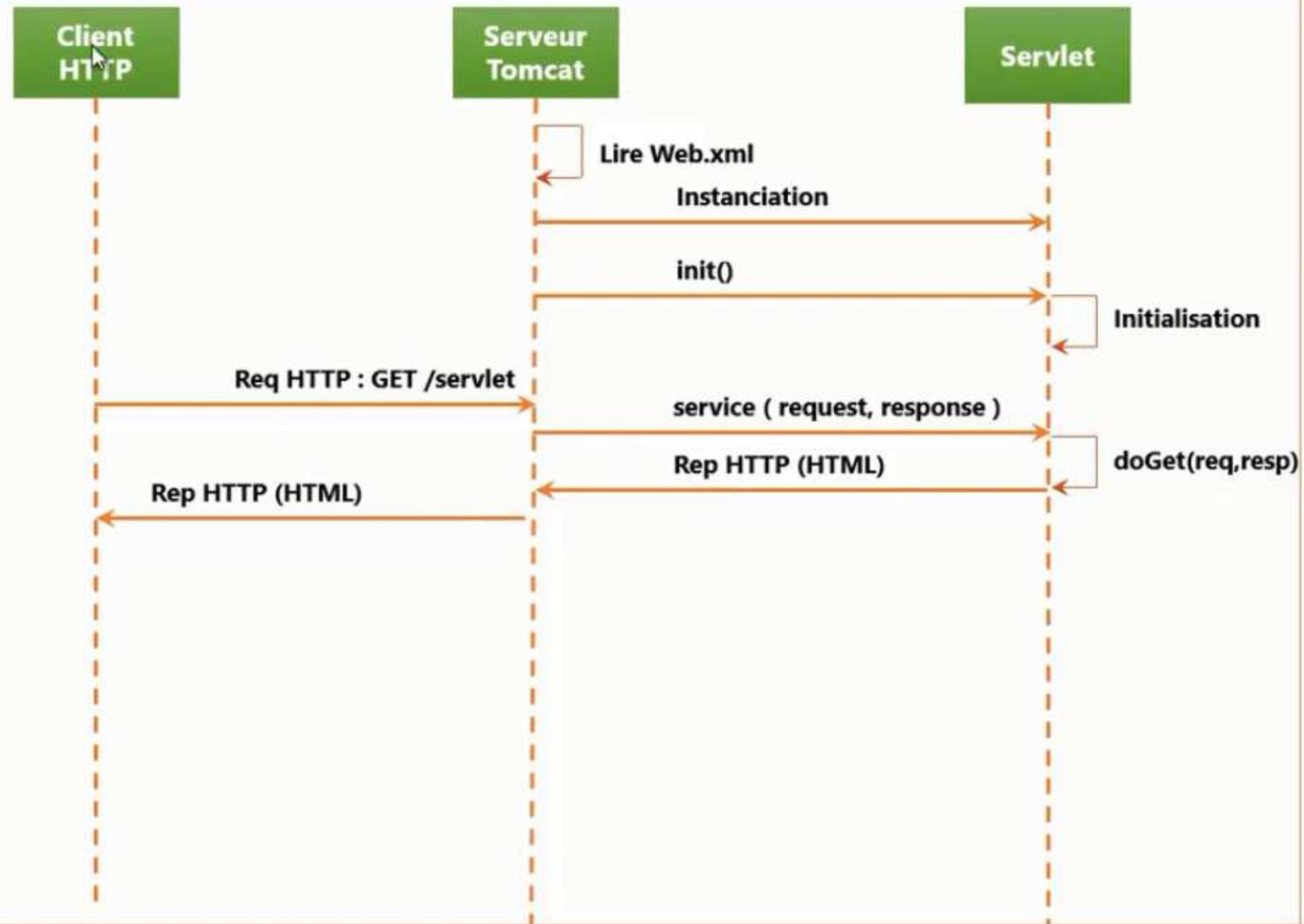
Servlet Life Cycle



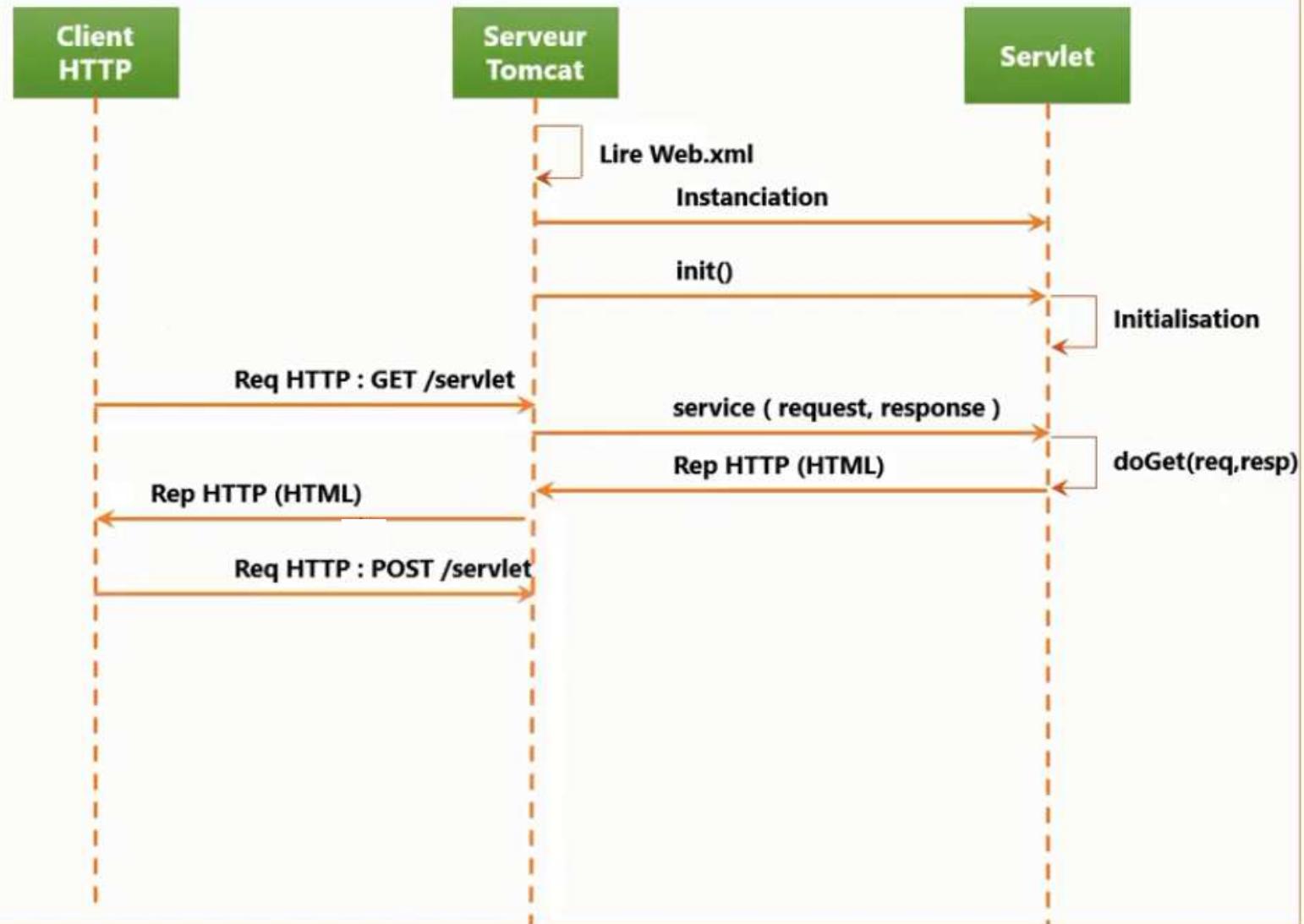
Servlet Life Cycle



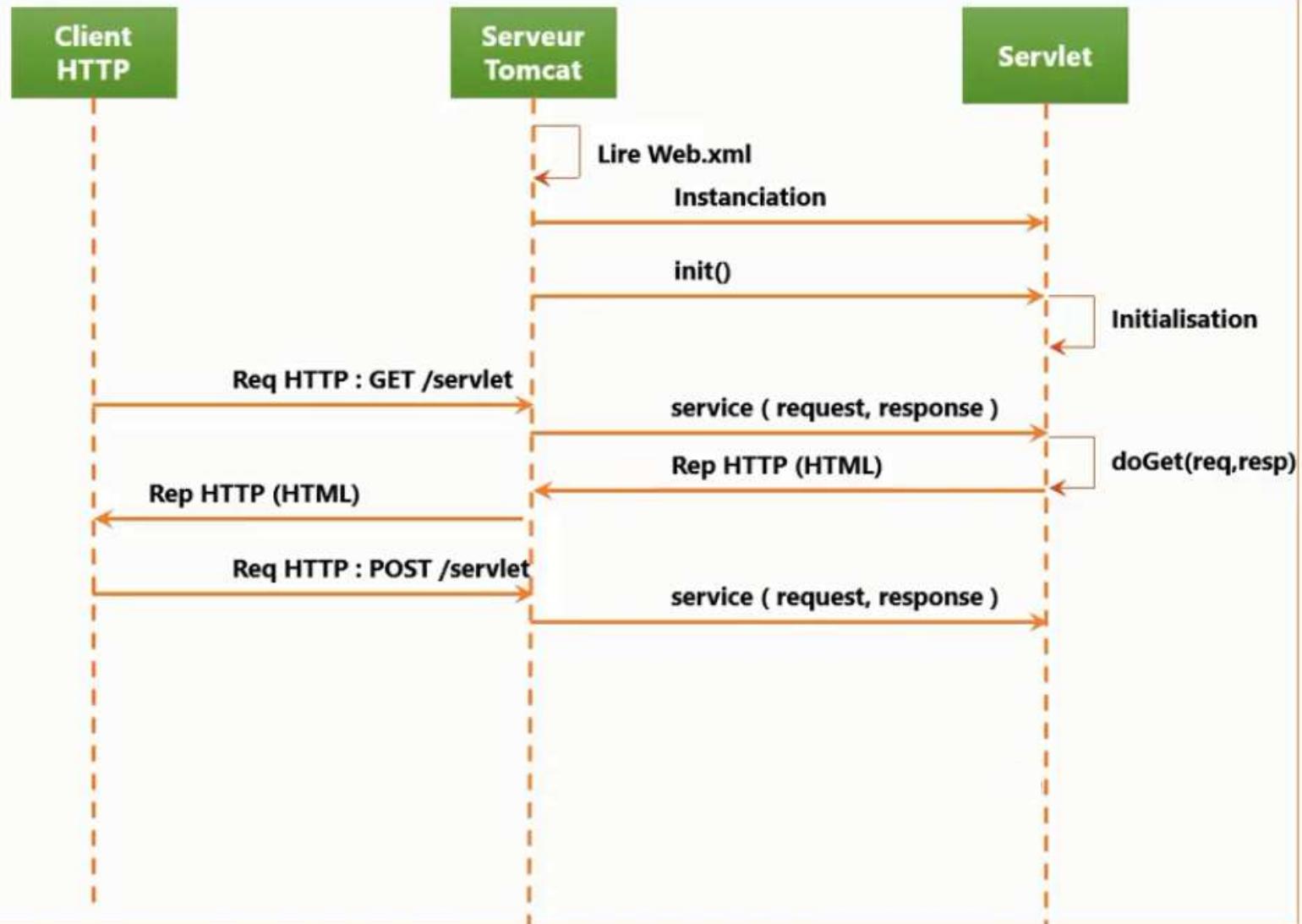
Servlet Life Cycle



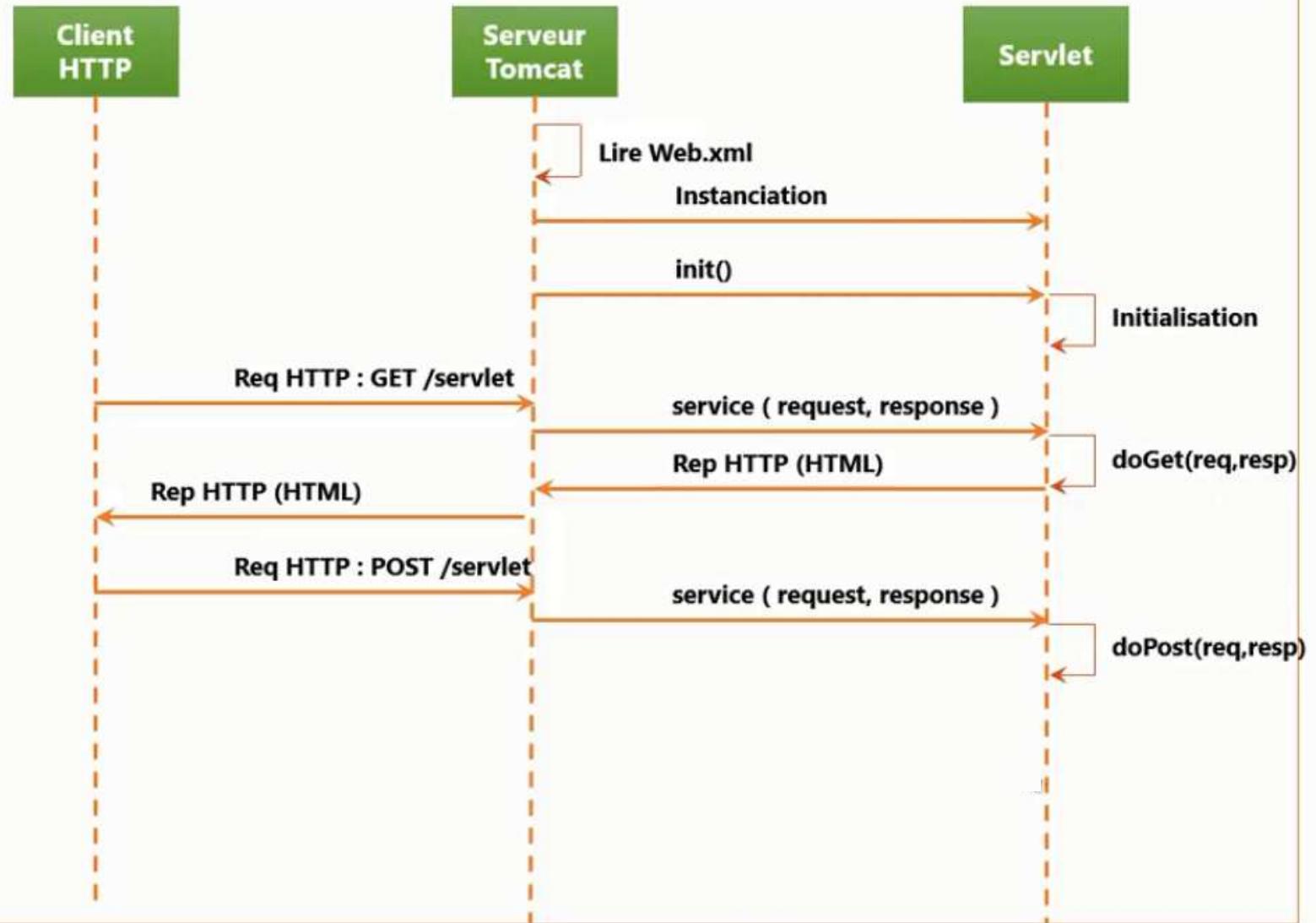
Servlet Life Cycle



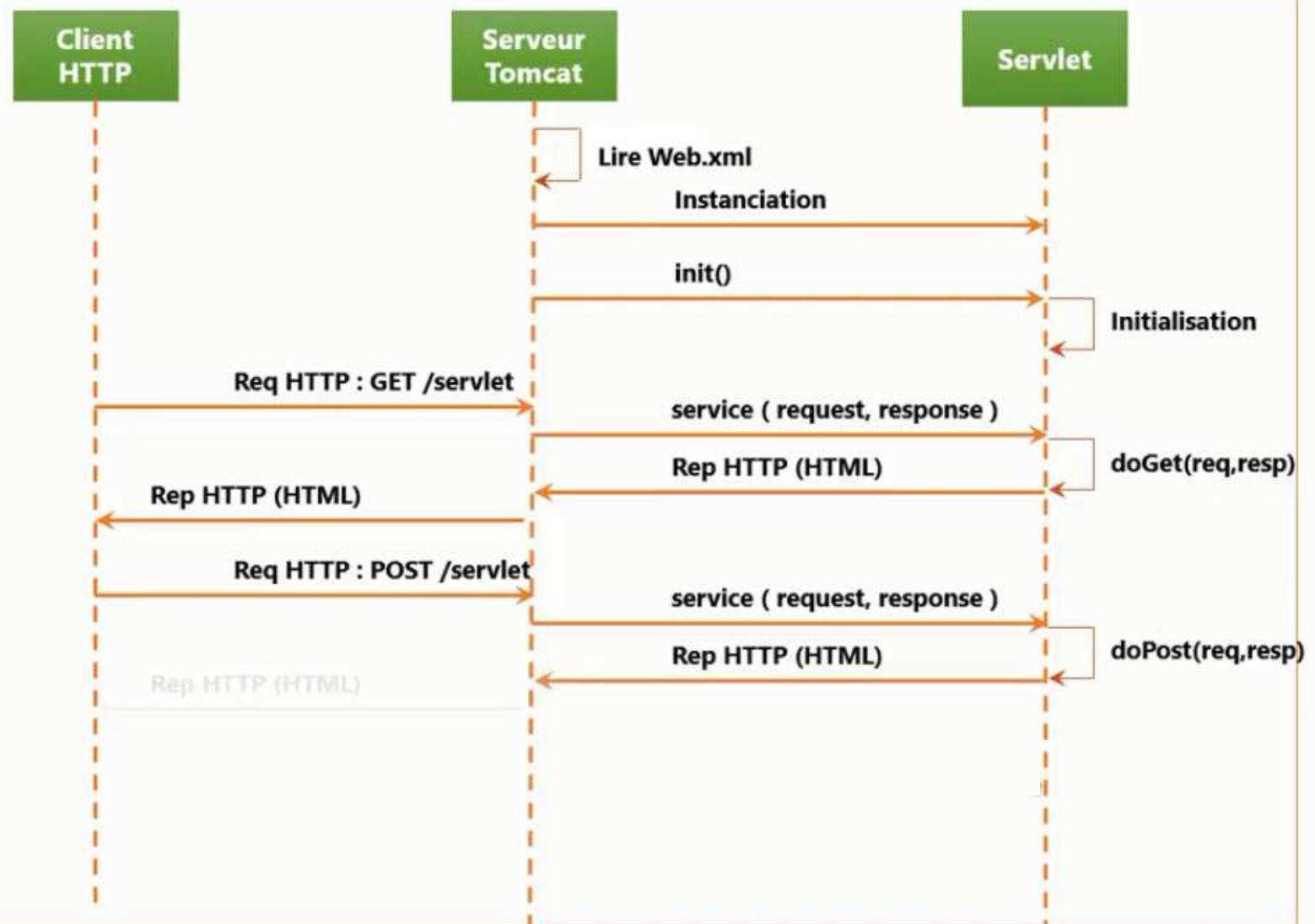
Servlet Life Cycle



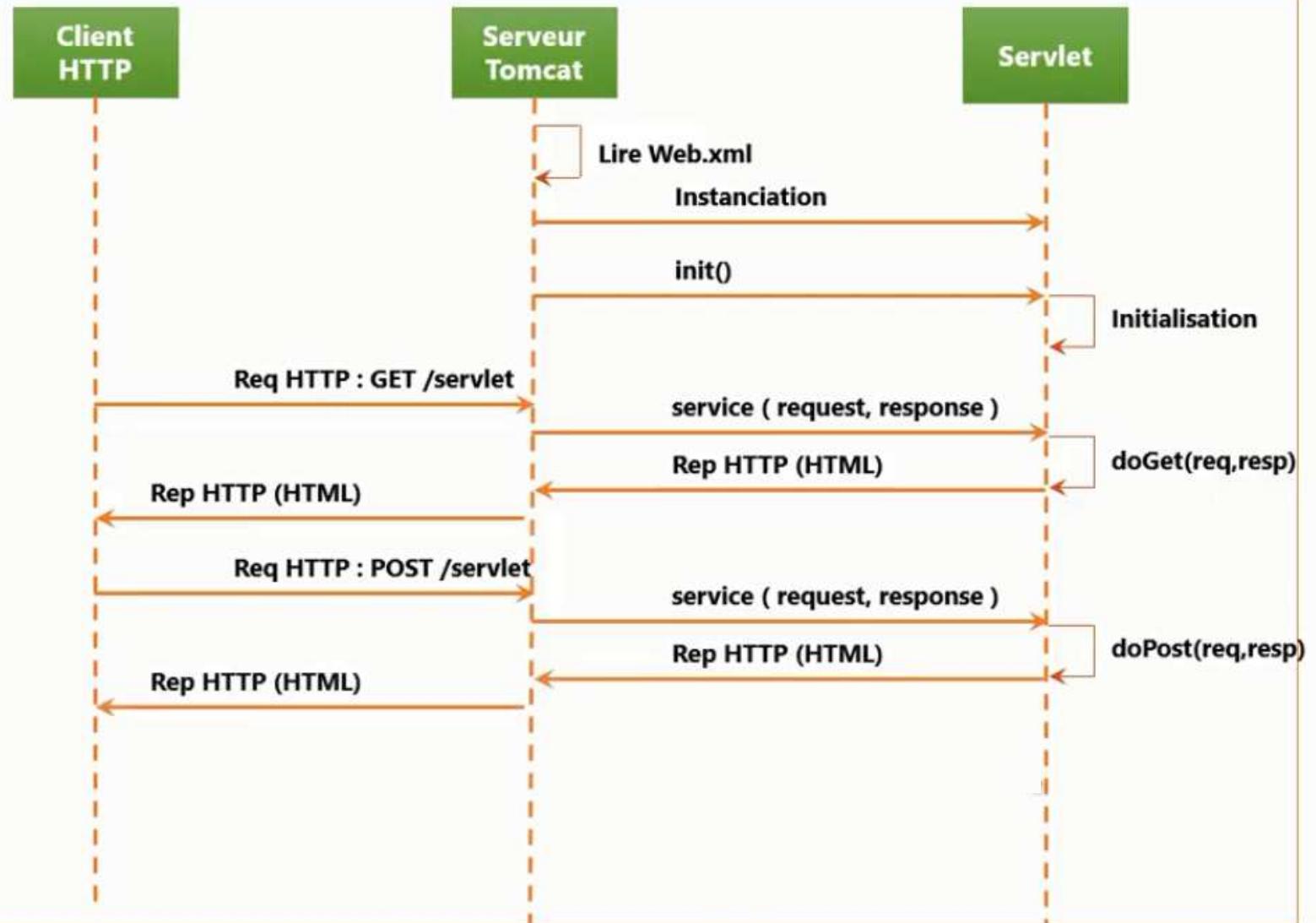
Servlet Life Cycle



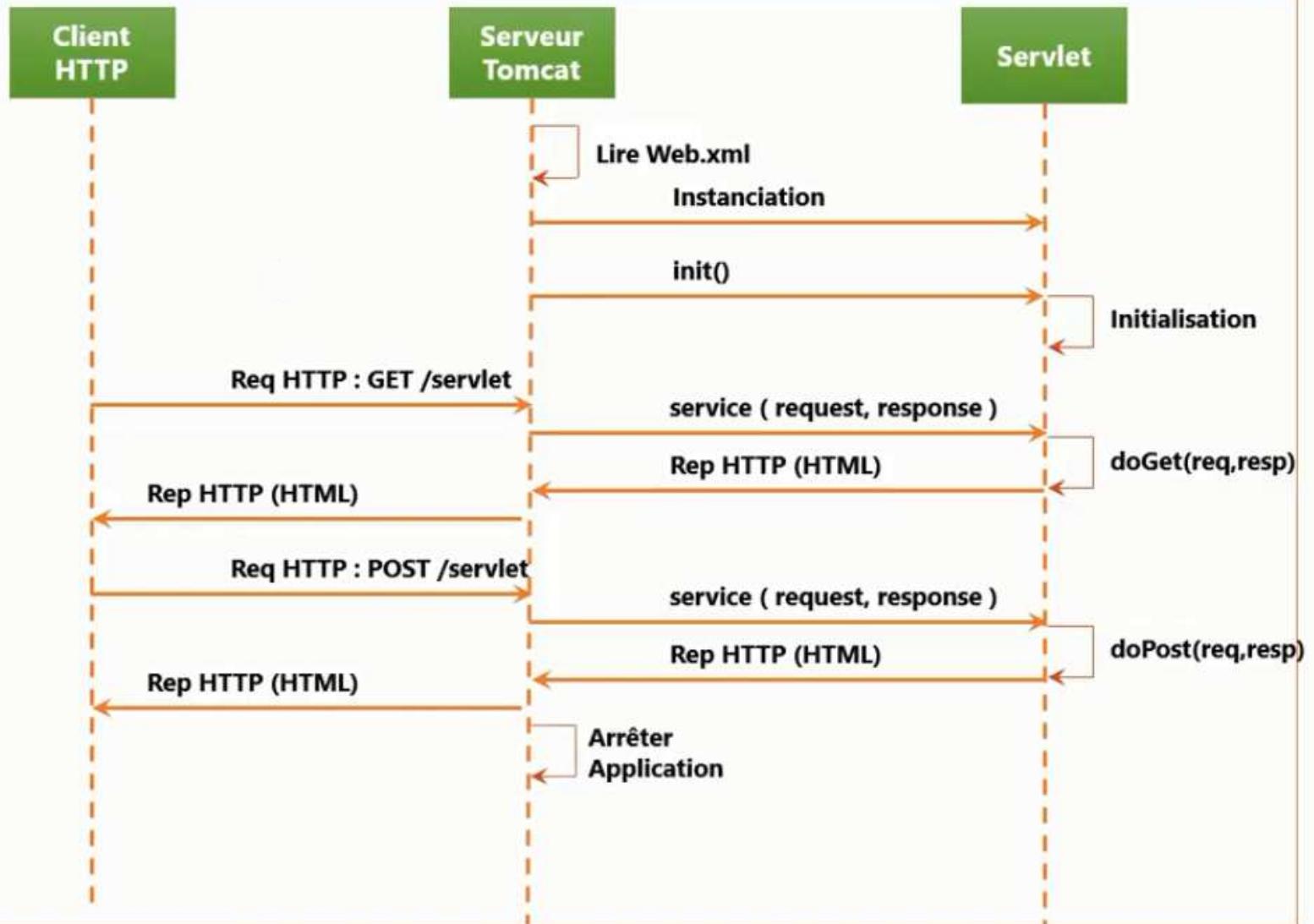
Servlet Life Cycle



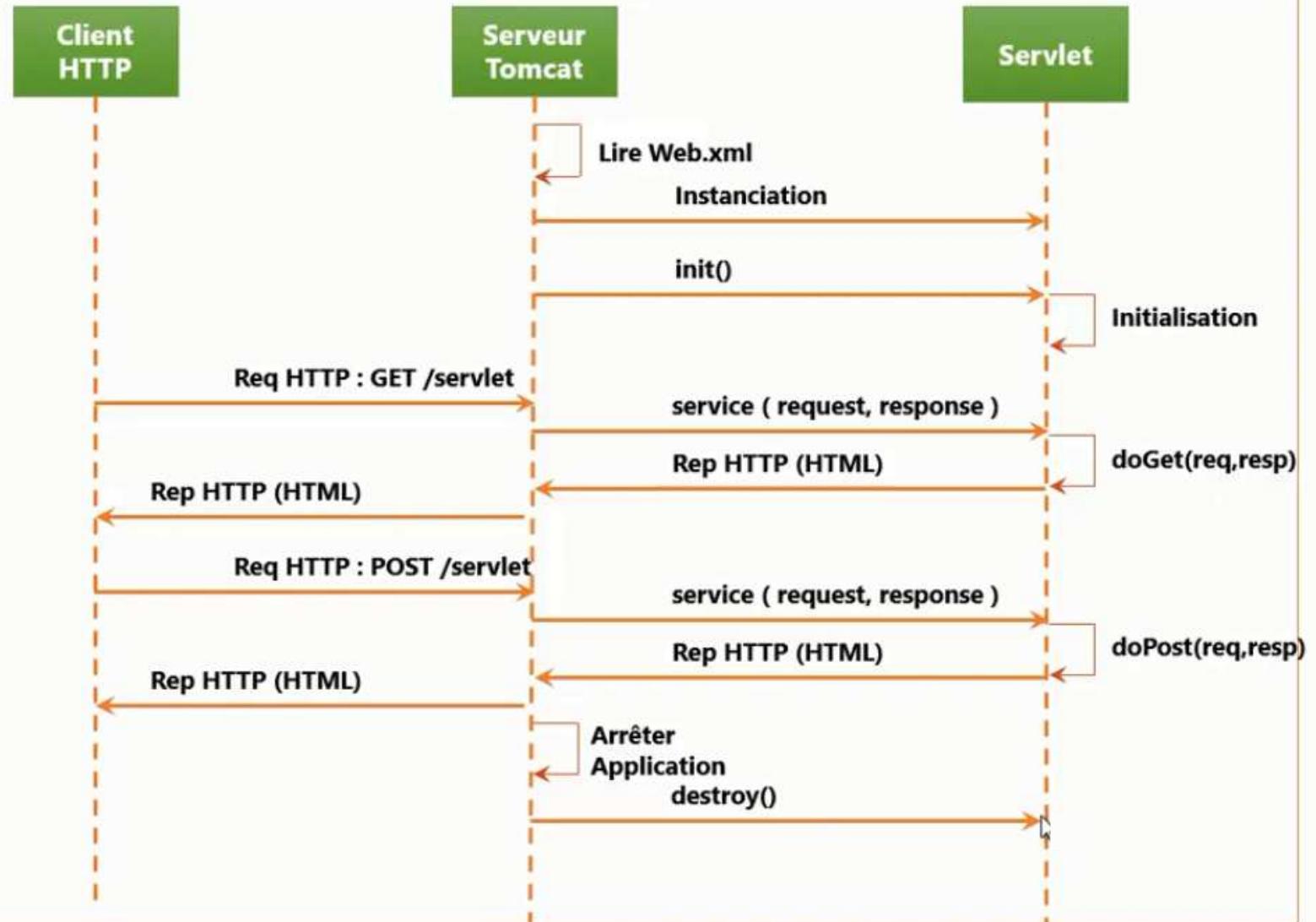
Servlet Life Cycle



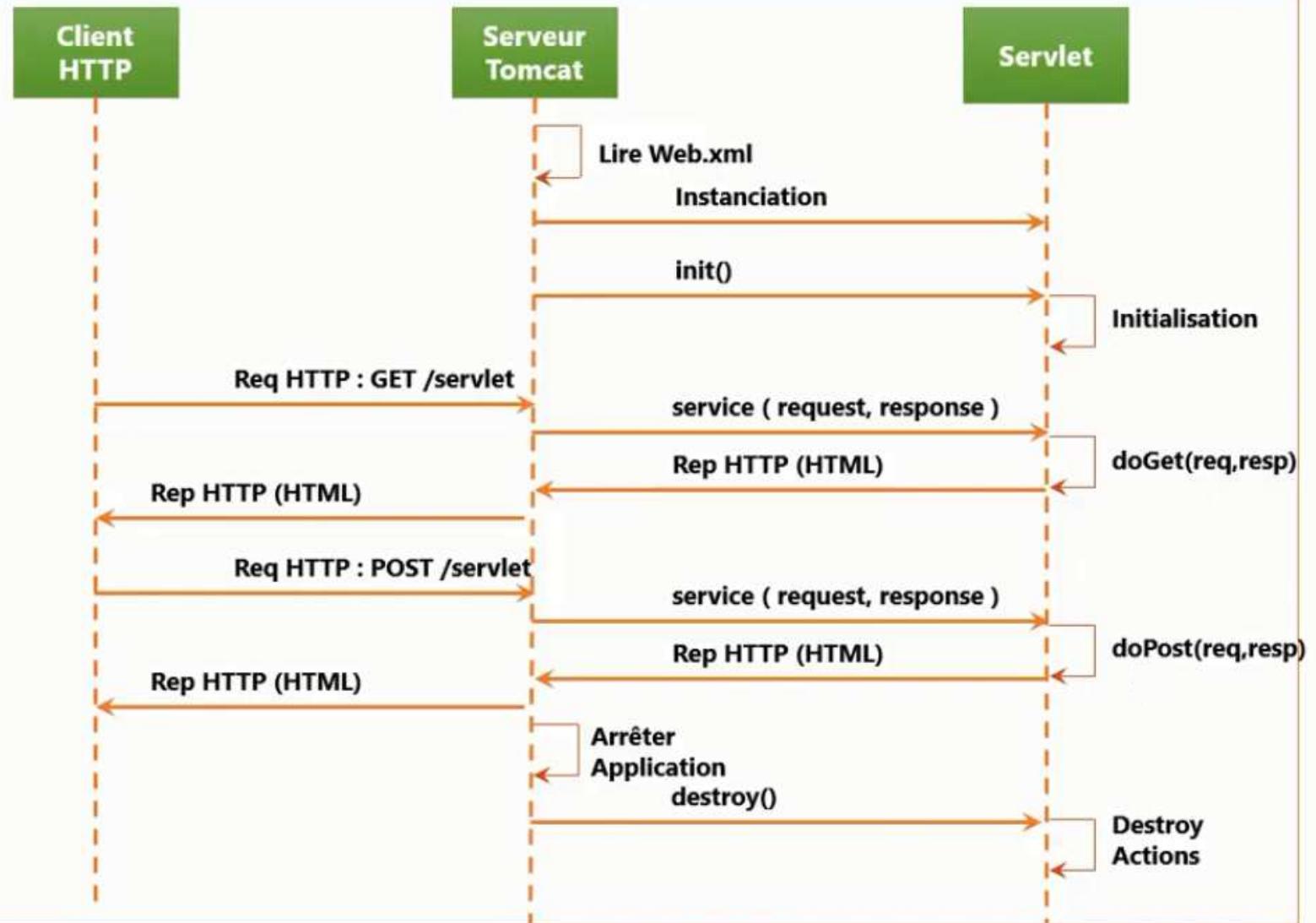
Servlet Life Cycle



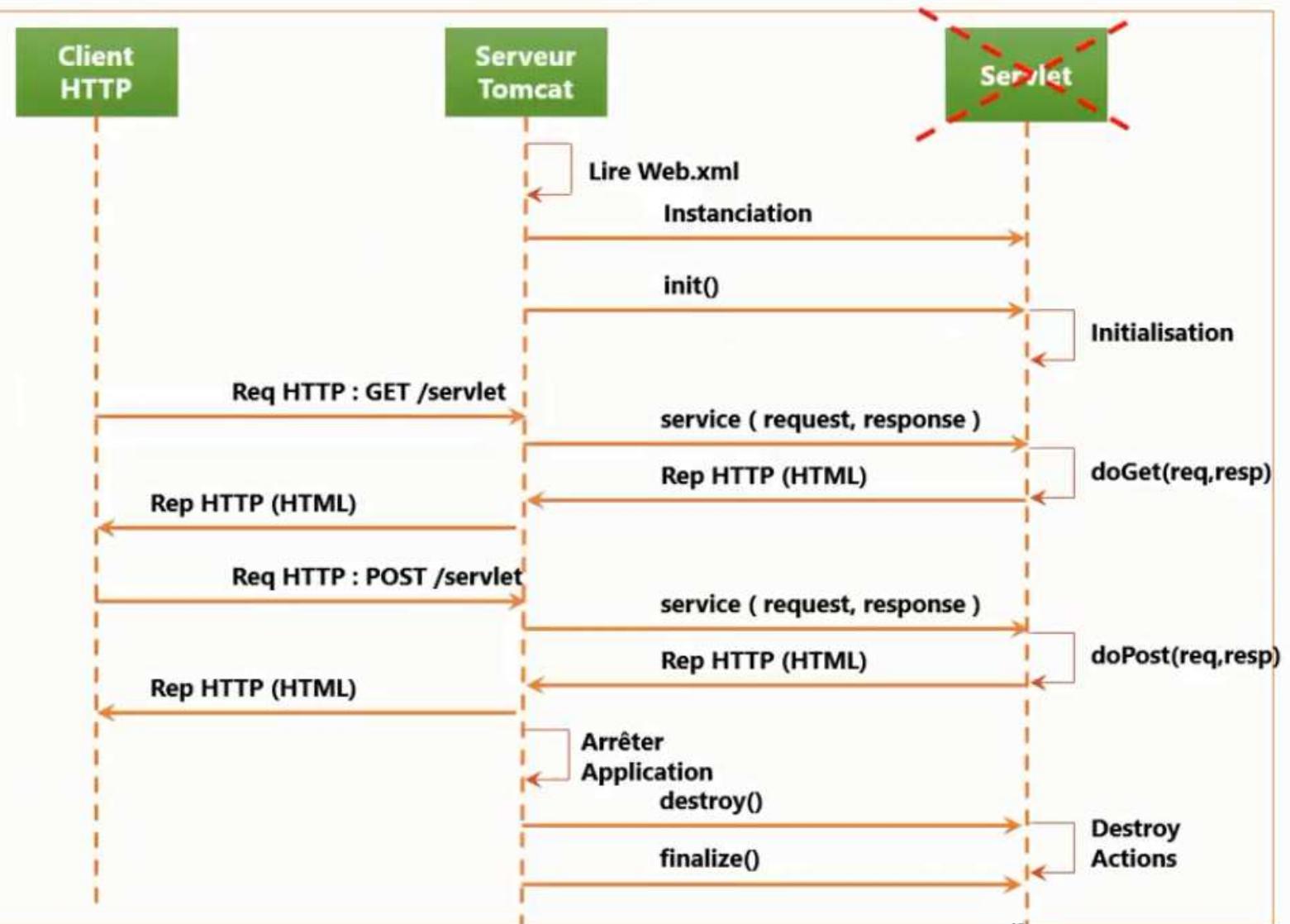
Servlet Life Cycle



Servlet Life Cycle

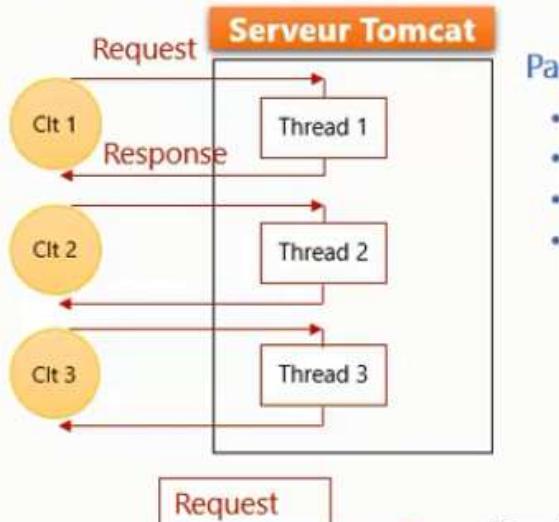


Servlet Life Cycle



Modèles : Multi Threads avec IO Bloquantes Vs Single Thread avec IO Non Bloquantes

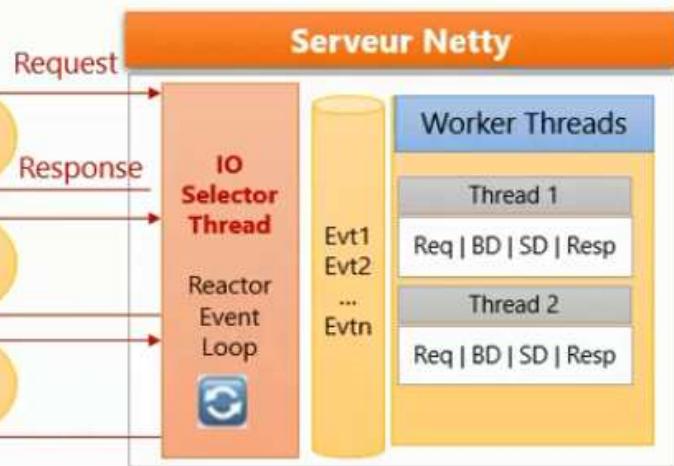
Multi Threads avec IO Bloquantes



Package java.io

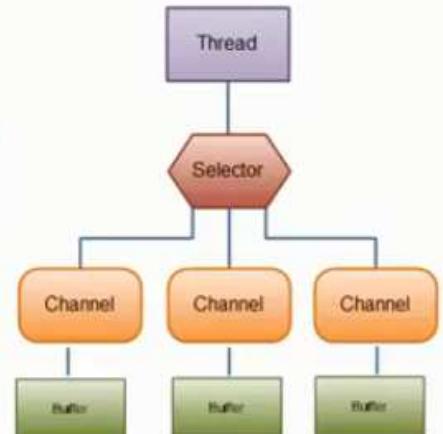
- InputStream
- OutputStream
- Reader
- Writer

Multi Single Thread avec IO Non Bloquantes



Package java.nio

- Channels:
 - SocketChannel
 - DatagramChannel
- Buffers
- Selector



Réactive Spring ou Spring Web Flux avec Netty

Projet Web Dynamique avec Eclipse

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: TPS

Project location
 Use default location
Location: C:\Docs\WS\SC8\TPS

Target runtime
<None>

Dynamic web module version
3.0

Configuration
Default Configuration

The default configuration provides a good starting point. Additional facets can later be installed for more functionality to the project.

New Server Runtime Environment

New Server Runtime Environment
Define a new server runtime environment

Select the type of runtime environment:
type filter text

Apache

- Apache Tomcat v3.2
- Apache Tomcat v4.0
- Apache Tomcat v4.1
- Apache Tomcat v5.0
- Apache Tomcat v5.5
- Apache Tomcat v6.0
- Apache Tomcat v7.0
- Apache Tomcat v8.0
- Apache Tomcat v8.5**

Apache Tomcat v8.5 supports J2EE 1.2, 1.4, 1.5, 1.6, 1.7, 1.8

Create a new local server

Tomcat Server
Specify the installation directory

Name: Apache Tomcat v8.5

Tomcat installation directory: C:\Tools\apache-tomcat-8.5.28

JRE: jdk1.8.0_151

New Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: TPS

Project location
 Use default location
Location: C:\Docs\WS\SC8\TPS

Target runtime
Apache Tomcat v8.5

Dynamic web module version
3.1

Configuration
Default Configuration for Apache Tomcat v8.5

A good starting point for working with Apache Tomcat v8.5 runtime. Additional facets can later be installed for more functionality to the project.

Projet Web Dynamique avec Eclipse

The screenshot shows the Eclipse IDE interface for creating a new Dynamic Web Project. Two configuration dialog boxes are displayed side-by-side.

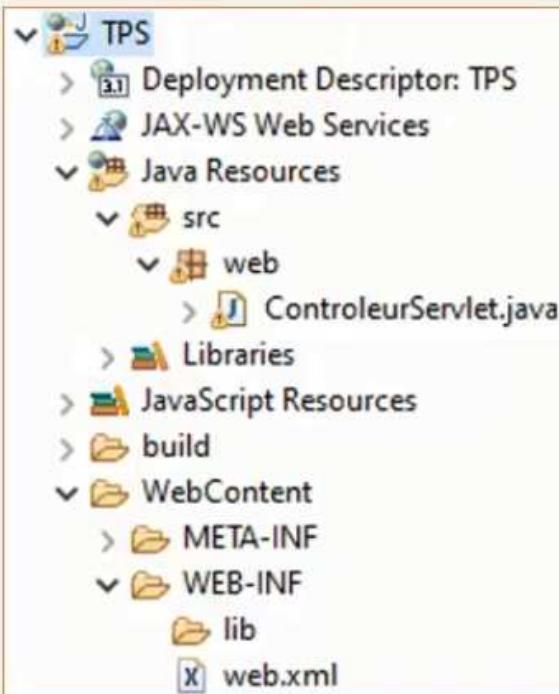
Java Configuration Dialog:

- Title:** New Dynamic Web Project
- Section:** Java
- Description:** Configure project for building a Java application.
- Source folders on build path:** A list containing "src". Buttons for "Add Folder...", "Edit...", and "Remove..." are available.
- Default output folder:** build\classes
- Buttons:** ? (Help), < Back, Next >, Finish (highlighted in blue), Cancel.

Web Module Configuration Dialog:

- Title:** New Dynamic Web Project
- Section:** Web Module
- Description:** Configure web module settings.
- Context root:** TPS
- Content directory:** WebContent
- Checkboxes:** Generate web.xml deployment descriptor (checked).
- Buttons:** ? (Help), < Back, Next >, Finish (highlighted in blue), Cancel.

Structure d'un projet Web Dynamique



- Le dossier src contient les classes java
- Le byte code est placé dans le dossier build/classes
- Les dossier WebContent contient les documents Web comme les pages HTML, JSP, Images, Java Script, CSS ...
- Le dossier WEB-INF contient les descripteurs de déploiement comme web.xml
- Le dossier lib permet de stocker les bibliothèques de classes java (Fichiers.jar)

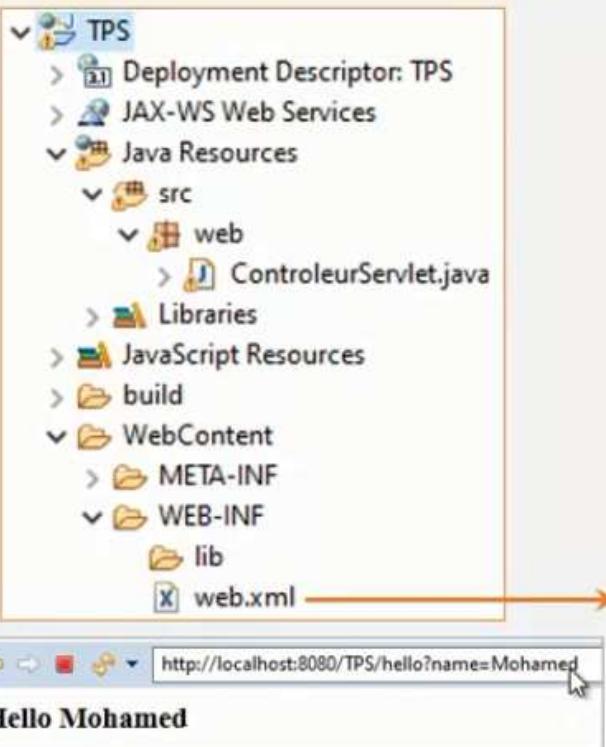
Déployer un Servlet

Pour que le serveur Tomcat reconnaise une servlet, celle-ci doit être déclarée dans le fichier web.xml qui se trouve dans le dossier WEB-INF.

Le fichier web.xml s'appelle le descripteur de déploiement de Servlet.

Ce descripteur doit déclarer principalement les éléments suivant :

- Le nom attribué à cette servlet
- La classe de la servlet
- Le nom URL à utiliser pour faire appel à cette servlet via le protocole HTTP.



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
  version="3.1">
  <display-name>TPS</display-name>

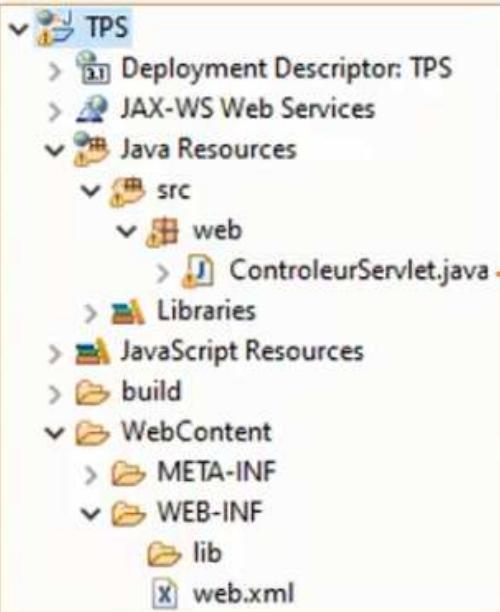
  <servlet>
    <servlet-name>cs</servlet-name>
    <servlet-class>web.ControleurServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>cs</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

Déployer un Servlet : Annotation @WebServlet

- Pour un projet web J2EE, utilisant un module web, version 3.0, le fichier web.xml n'est pas nécessaire.
- Dans ce cas, le déploiement d'une servlet peut se faire en utilisant l'annotation **@WebServlet**:



```
package web;  
import java.io.*; import javax.servlet.*;  
import javax.servlet.annotation.*;  
import javax.servlet.http.*;  
@WebServlet(name="cs",urlPatterns={"/hello","*.do"})  
public class ControleurServlet extends HttpServlet {  
}
```

Exemple de Servlet

```
package web;  
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;  
public class ControleurServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
    IOException {  
        response.setContentType("text/html");  
        PrintWriter out=response.getWriter();  
        String name=request.getParameter("name");  
        out.println("<html><head><title>Hello Servlet</title></head>");  
        out.println("<body>");  
        out.println("<h3>Hello "+name+"</h3>");  
        out.println("</body></html>");  
    }  
}
```



Servlet Vs JSP (Java Server Pages)

Servlet

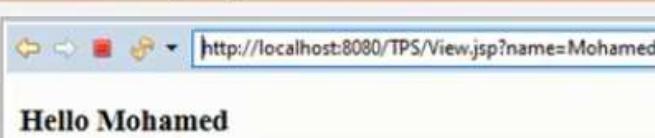
```
package web; import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String name=request.getParameter("name");
        out.println("<html><head><title>Hello
Servlet</title></head>");
        out.println("<body>");
        out.println("<h3>Hello "+name+"</h3>");
        out.println("</body></html>");
    }
}
```



View.jsp

```
<%
String name=request.getParameter("name");
%>
<!DOCTYPE html>
<html>
<head>
    <title>Hello JSP</title>
</head>
<body>
    <h3>Hello <%=name%></h3>
</body>
</html>
```

- Servlet : Classe Java dans laquelle on génère du code HTML
- JSP : Forme d'une page HTML dans laquelle on écrit du code Java
- Quand une JSP est appelée la première fois, elle sera convertie en Servlet par Tomcat.
- Dans une application web JEE, on utilise les deux :
 - Servlet pour jouer le rôle du contrôleur
 - JSP pour jouer le rôle d'une Vue



Communication entre Servlet et JSP : Forward



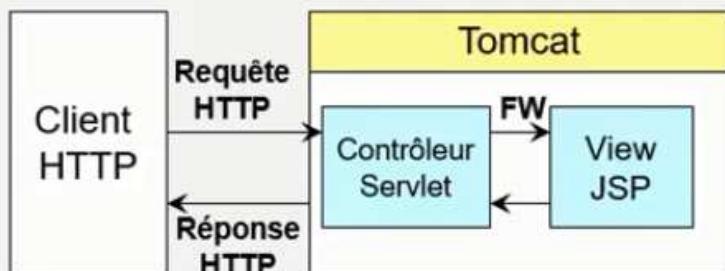
Pour séparer les rôles une servlet peut faire un forward vers une JSP de la manière suivante :

```
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        String name=request.getParameter("name");
        request.setAttribute("inputData", name);
        request.getRequestDispatcher("View.jsp").forward(request, response);

    }
}
```

```
<%>
String name=(String)request.getAttribute("inputData");
<%>
<!DOCTYPE html>
<html>
<head>
<title>Hello JSP</title>
</head>
<body>
<h3>Hello <%=name%> From JSP</h3>
</body>
</html>
```



Communication entre Servlet et JSP : Redirection

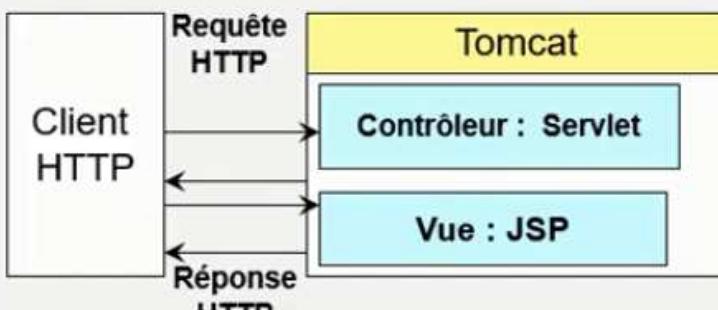
```
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {

        String name=request.getParameter("name");
        response.sendRedirect("View.jsp?name="+name);
    }
}
```



http://localhost:8080/TPS/View.jsp?name=Mohamed

Hello Mohamed From JSP



Client HTTP

Requête HTTP

Tomcat

Contrôleur : Servlet

Réponse HTTP

Vue : JSP

```
<%>
String name=request.getParameter("name");
%>
<!DOCTYPE html>
<html>
<head>
<title>Hello JSP</title>
</head>
<body>
<h3>Hello <%=name%> From JSP</h3>
</body>
</html>
```

