Découverte de JQuery et AJAX

C. BENSARI

JQuery

- JQuery est une librairie JS qui permet l'écriture du code JavaScript grâce à l'utilisation de nombreuses méthodes
- Le slogan de JQuery est : *write less, do more*
- Pour utiliser JQuery , il faut importer une version de la librairie jquery depuis le site officiel et la déclarer en utilisant la balise <script></script> dans le document Html
- Pour utiliser JQuery dans JavaScript, ll faut utiliser le symbole $ qui va référencer l'objet jQuery

JavaScript natif (vanilla) vs JQuery

Code JavaScript natif :

```
var elements = document.getElementsByClassName('red-link')
for (var i = 0; i < elements.length; i++) {
  var element = elements[i]
  element.addEventListener('click', function (e) {
    e.preventDefault()
    alert("Un lien qui ne mène nul part !")
  })
  element.style.color = "#FF0000"
}
```

Code JQuery :

```
$('.red-link').click(function (e) {
  e.preventDefault()
  alert('Un lien qui ne mène nul part !')
}).css('color', '#FF0000')
```

JavaScript natif (vanilla) vs JQuery

code JavaScript natif:

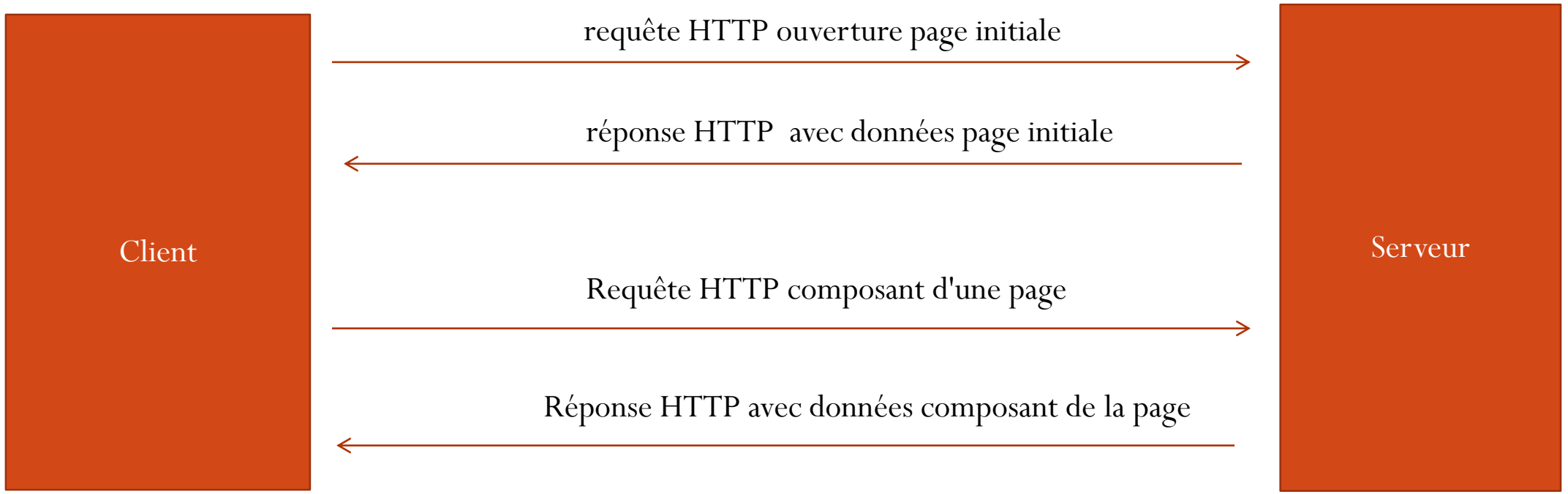
```
var cls = 'active'
var elt = document.getElementById('#menu')
if (elt.className.length == 0){
  elt.className = cls;
} else if (elt.className.indexOf(cls) < 0) {
  elt.className += ' ' + cls;
}
```

code JQuery :

```
$('#menu').addClass('active')
```

AJAX : Asynchronous JavaScript And Xml

- Ajax est une technique permettant de récupérer/modifier des données (en format JSON ou XML, …) **depuis le serveur** du back-end de manière **asynchrone**
- Il permet ainsi, la mise à jour d'une partie du contenu de la page web **sans** la recharger entièrement



XML : eXtensible Markup Langage

- XML est un langage de balisage, il utilise le même principe que celui de Html
- A la différence de HTML, avec XML les balises ne sont pas prédéfinies mais c'est plutôt à l'utilisateur de les créer lui-même
- XML est un format de données utilisé pour les échanges entre machines et applications
- Exemple d'un fichier ***.xml :

XML : eXtensible Markup Langage

```
<?xml version=”1.0” ?>
<bibliotheque>
   <livre pages= ”250” titre= ”Les misérables”>
         <auteur>Victor HUGO</auteur>
         <isbn>13966662</isbn>
   </livre>
   <livre pages= ”320” titre= ”L’instant présent”>
         <auteur>Guillaume MUSO</auteur>
         <isbn>12586662</isbn>
   </livre>
</bibliotheque>
```

Le format JSON (JavaScript Object Notation)

- JSON est un format de représentation de données
- Il utilise le principe de correspondances clé-valeur
- Exemple d'un objet au format JSON en JS:

```
var personne = {
        “prenom” : “David”,
        “nom” : “DUPOND”,
        “hobies” : [“Tennis”, “Natation”]
};
var personneHobies = personne.hobies;
```

Implémentation d'un appel Ajax en JS (ES5)

```
<script>
    var objXMLHttpRequest = new XMLHttpRequest();
    objXMLHttpRequest.onreadystatechange = function() {
      if(objXMLHttpRequest.readyState === 4) {
        if(objXMLHttpRequest.status === 200) {
              alert(objXMLHttpRequest.responseText);
        } else {
              alert('Error Code: ' +  objXMLHttpRequest.status);
              alert('Error Message: ' + objXMLHttpRequest.statusText);
        }
      }
    }
    objXMLHttpRequest.open('GET', 'request_ajax_data.php');
    objXMLHttpRequest.send();
</script>
```

Implémentation d'un appel Ajax en JS

1- Initialiser l'objet **XMLHttpRequest** responsable des appels AJAX

2- L'objet **XMLHttpRequest** possède la propriété **readyState**. La valeur de cette propriété change durant le cycle de vie de la requête. Elle prend l'une des valeurs suivantes : *OPENED*(1),*HEADERS_RECEIVED*(2), *LOADING*(3) et *DONE*(4)

3- Ajout d'un listener d'événement en utilisant la propriété **onreadystatechange** et l'associer à une fonction qui sera appelée à chaque changement du statut de la requête

4- La fonction vérifie en premier si le readyState a la valeur 4 (*DONE*) qui veut dire : "*requête terminée et que nous avons reçu une réponse du serveur*". La fonction vérifie ensuite si le **status de la réponse** est égale à 200 (requête réussie) et à la fin affiche le contenu texte de la réponse via la propriété responseText

Implémentation d'un appel Ajax en JS

5- Aprés avoir mis en place le listener, on initialise la requête en utilisant la methode **open** de l'objet **XMLHttpRequest**. Après l'appel de la fonction open, la valeur de l'attribut **readyState** sera égale à 1 (*OPENED*). La méthode open prend deux paramètres : la méthode HTTP (GET,POST, PUT,..) et le chemin vers le programme du back-end

6- Appel de la fonction **send** de l'objet **XMLHttpRequest** qui renvoie la requête vers le serveur ce qui fait passer readyState à 2 (HEADERS_RECEIVED)

7- Une fois le serveur répond à la requête, readyState passe à la valeur 4 (DONE) et une fenêtre d'alerte s'affiche avec le message envoyé par le serveur

Implémentation d'un appel Ajax en JQuery

```
<script>
    $.ajax(
      'request_ajax_data.php',
      {
          success: function(data) {
            alert('Appel AJAX réussi !');
            alert('Données reçues du serveur : ' + data);
          },
          error: function() {
            alert('Erreur détectée lors de l''appel AJAX');
          }
       }
    );
</script>
```

Implémentation d'un appel Ajax en JQuery

1- Utilisation du symbôle $ pour référencer l'objet jQuery

2- La méthode ajax prend une URL comme premier paramètre qui sera appelée en arrière plan pour récupérer/envoyer du contenu depuis le serveur

3- Le deuxième paramètre est un objet JS, cet objet peut avoir différents attributs/propriétés pour ajouter des options aux appels AJAX

4- Dans notre cas (et la plupart des cas) nous utilisons les attributs success et error (callbacks). Le callback success est appelé lorsque l'appel AJAX réussi et la réponse du serveur passera en paramètre de la fonction (data). Le callback error est appelé lorsque la requête ou le serveur rencontre un problème pendant l'éxécution de la requête AJAX

JQuery: utilisation de promise et deferred

- Gràce à AJAX il est possible d'effectuer des appels asynchrones en arrière plan
- L'inconvénient des appels asynchrones c'est que lorsque un appel AJAX est fait, et juste après cet appel un traitement est effectué sur le résultat de la requête AJAX
- Dans ce cas de figure, la requête AJAX est envoyée au serveur et « immédiatement » après, le traitement sur le résultat est exécuté (sans être sûre que la requête AJAX a terminé). Ce qui mène vers une erreur
- Pour remédier à ce problème, la première solution est d'effectuer le traitement dans le callback success de la requête AJAX
- Mais que faut il faire si un autre appel AJAX est effectué avec comme paramètres, le résultat du premier appel ? ➔ promise & deferred

JQuery: utilisation de promises et deferred

- **promises** est un paradigme de programmation qui gére les problèmes des appels asynchrones. La gestion est faite en différant (**Deferred**) l'exécution d'un traitement jusqu'à ce qu'un autre traitement termine son exécution :

```
var p = fonction1();
p.done(function(result) {
    fonction2();
});
```

- Dans cet exemple, fonction1 (asynchrone) retourne un objet **promise** qu'on affecte à une variable appelée "p" par convention
- Promise est un objet qui possède deux fonctions de callback (done et fail). Dans cet exemple, le callback done est utilisé, et il sera exécuté après la fin d'exécution de fonction1 avec succès

JQuery: utilisation de promises et deferred

- Pour créer un objet **promise**, il faut au préalable créer un objet **Deferred** qui nous permettra l'accès à un **promise**

```
function fonction1(){
    var d = $.Deferred();

    // .....

    return d.promise();
}
```

- Dans cet exemple, le promise est créé mais n'informe pas comment le traiatment a terminé (success ou erreur)
- Pour ajouter cette information **Deffered** possède deux fonctions (resolve() et reject())

JQuery: utilisation de promises et deferred

- Exemple d'utilisation de resolve et reject :

```
var p = fonction1('traitement.php');
p.done(function(resultat) {
    // utilisation du résultat de la requête
    // passé en paramètre du resolve
    var data = resultat;
    console.log(data);
});
p.fail(function(resultat) {
// resultat est le string passé à la fonction reject!
var error = resultat;
console.log(error);
});
```

```
function fonction1(url) {
    var d = $.Deferred();
    $.ajax({
      url: url,
      success: function(resultat){
          d.resolve(resultat);
      },
      error: function(){
          d.reject("Erreur !");
      }
    })
    return d.promise();
```