Ajax

Cyril Rabat, François Alin

Programmation Web - Php

2019-2020

Cours Info 303

Présentation d'Ajax

Version 14 octobre 2019

Table des matières

Rappels sur *Javascript*

- Créé en 1995, standardisé sous ECMAScript en 1997
  ↪*Javascript* est une implémentation d'ECMAScript
- Langage de script exécuté au sein d'une page Web
  - Intégré dans le code HTML
  - Interprété par le navigateur
    ↪Attention cependant aux incompatibilités entre navigateurs !
- Basé en partie sur la syntaxe du *Java*
- Objectif : rendre la page dynamique
- Une API complète
- Différents composants/éléments/langages liés :
  - AJaX
  - JSON

Quelques mots sur le cache

- Rappels lors de l'accès à un URL :
  - Script principal chargé par le navigateur
  - Éléments secondaires (images, scripts, *etc.*) : utilisation du cache
- Scripts *Javascript* situés dans des fichiers séparés :
  ↪Évite la surcharge du code HTML
  ↪Libère de la bande passante (scripts non rechargés à chaque fois)
- Problèmes dans le cas de la programmation *Javascript* :
  ↪Modifications des scripts non prises en compte sur le client !
- Solutions :
  - Vider l'historique du navigateur
  - Inclure le *Javascript* dans le fichier HTML
    ↪Uniquement en mode développement
  - Configuration du serveur Web (empêche la mise en cache)
  - Manuellement avec un numéro de version :
    ↪`<script src="script.js?v=<?php echo time(); ?>" ...`

AJaX

- AJaX pour **A***synchronous* **Ja***vaScript and* **X**ML
- Exploiter *Javascript* pour l'envoi de requêtes HTTP
  ↪Récupération de données dans la page en cours
- Les réponses peuvent ensuite être analysées :
  ↪Généralement au format XML ou JSON
  ↪Peut être de tout type (exemples : JSON, HTML, *etc.*)
- Utilisation de la classe *Javascript* : `XMLHttpRequest`
- Du côté du serveur : aucun changement !

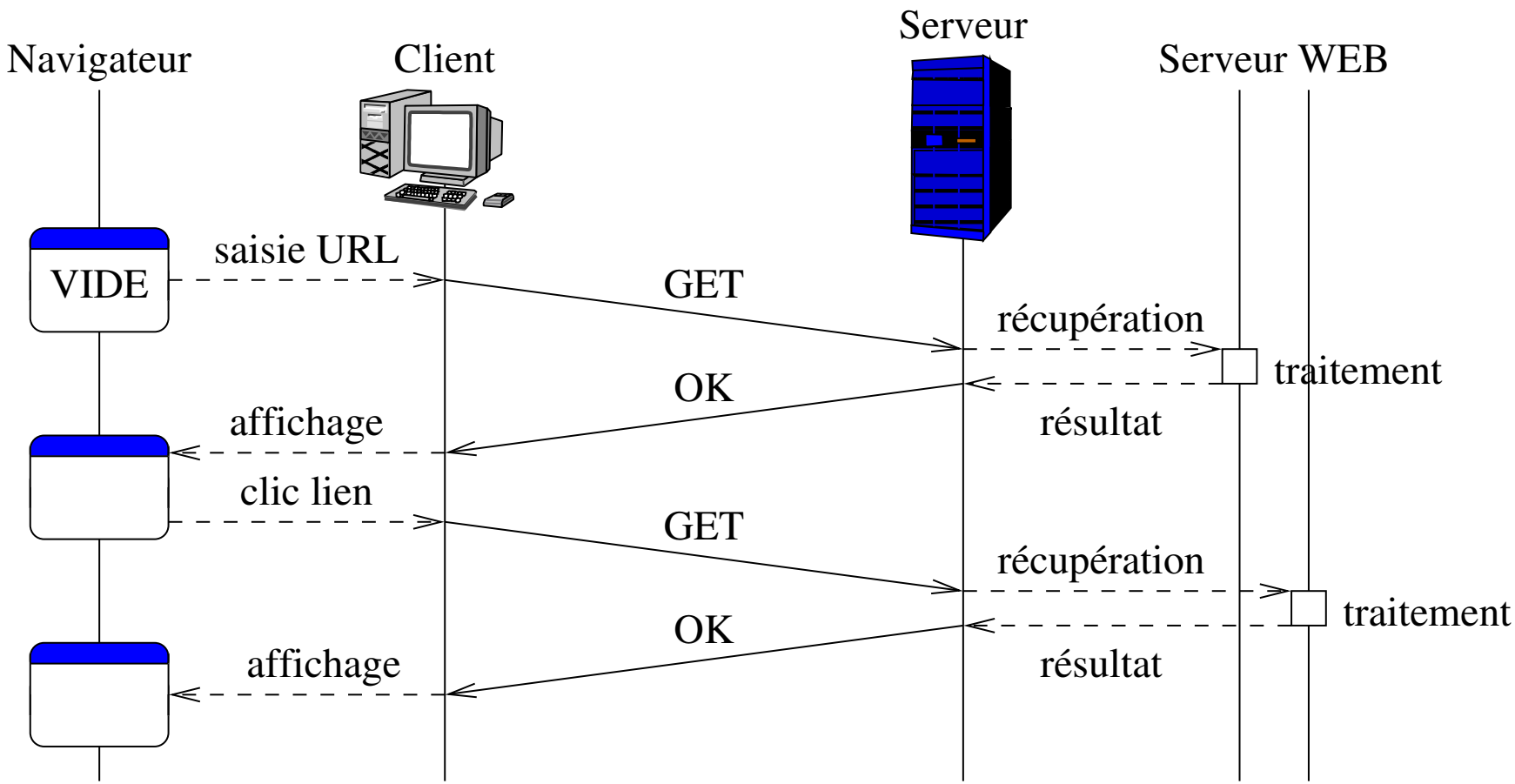
Fonctionnement d'AJaX

- Sans AJaX, à chaque requête HTTP, nouvelle page créée par le navigateur
- Avec AJaX, possibilité d'exécuter des requêtes au sein d'une même page Web :
  ↪Pas de rechargement de la page
- Intérêts :
  - Pages dynamiques (modification du DOM de la page)
  - Plus grande ergonomie pour l'utilisateur
  - Pas de nouveau langage (réutilisation de standards)
  - Données présentes sur le client et sur le serveur
- Les appels peuvent être asynchrones...
  ↪L'utilisateur garde la main pendant la réception des données
- ...ou synchrones

Limites d'AJaX

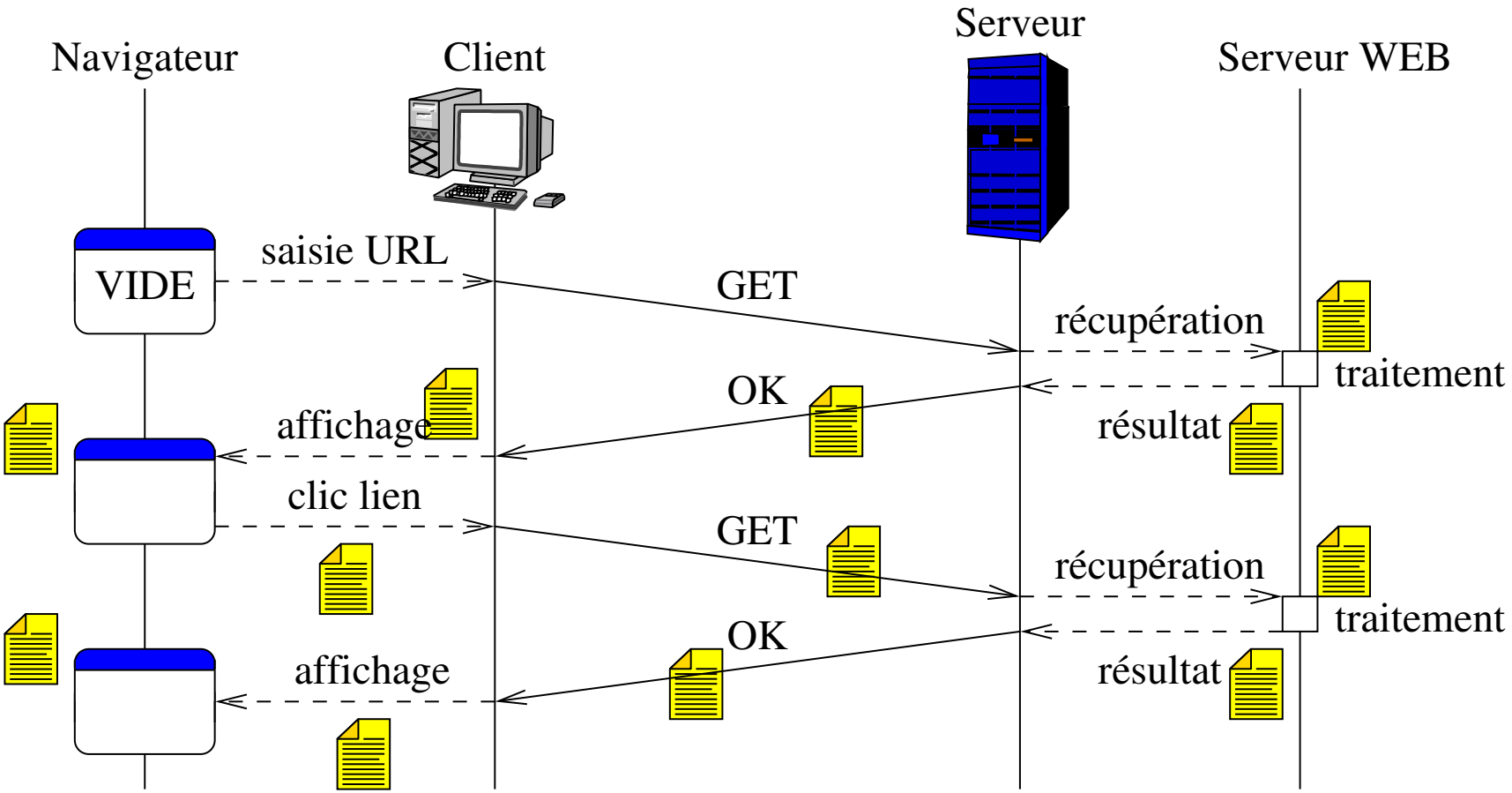
- Croisement de domaine :
  - Par mesure de sécurité, impossible de récupérer des données sur un autre domaine avec AJaX
  - Travail actuel du W3C, même si des implémentations (propriétaires) existent
- Techniquement, toutes requêtes HTTP autorisées
  ↪dont GET et POST
- Mais impossible d'envoyer des fichiers !
- Attention à l'encodage des données (UTF-8)

Différences d'appels



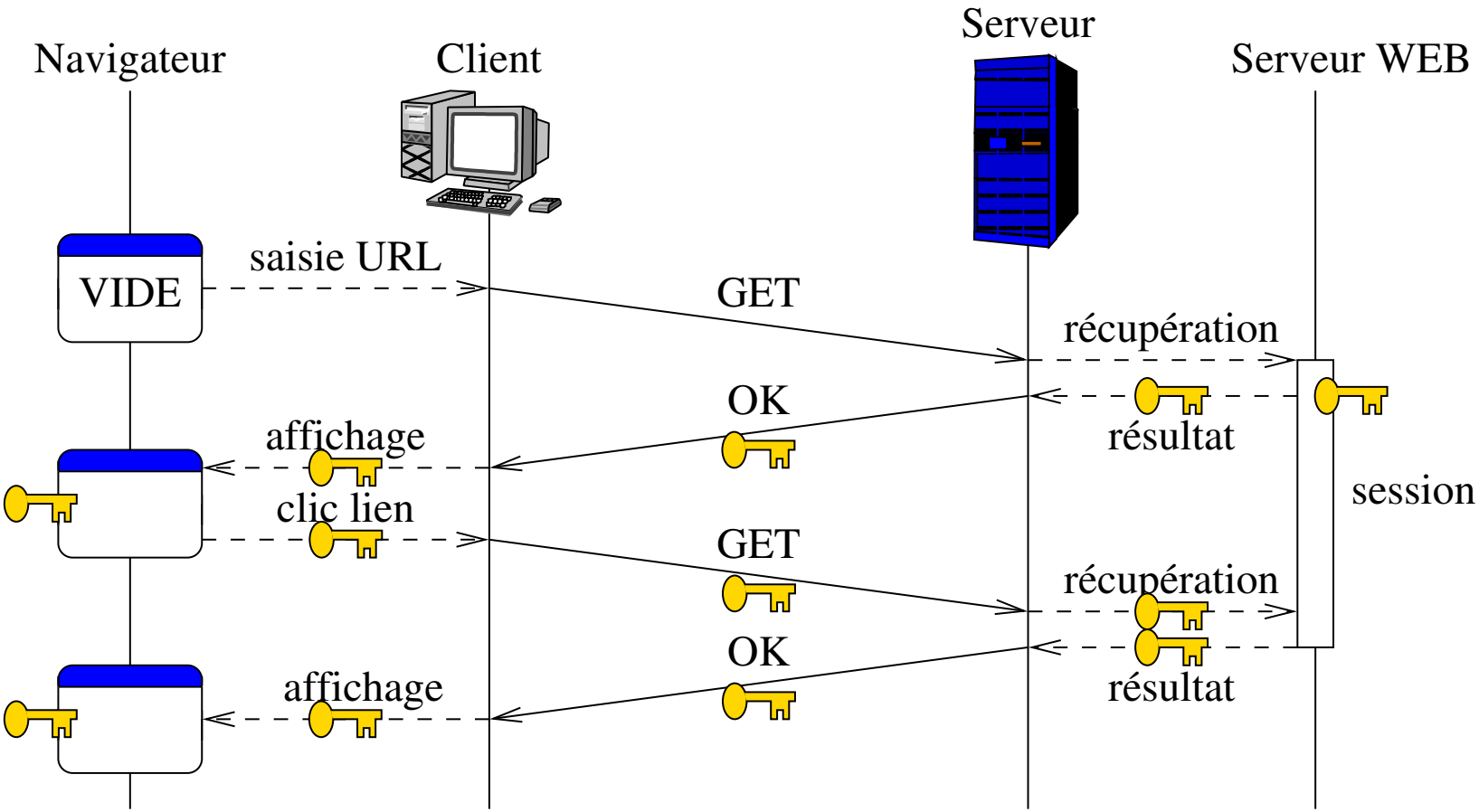
Série de requêtes/réponses HTTP classiques

Différences d'appels



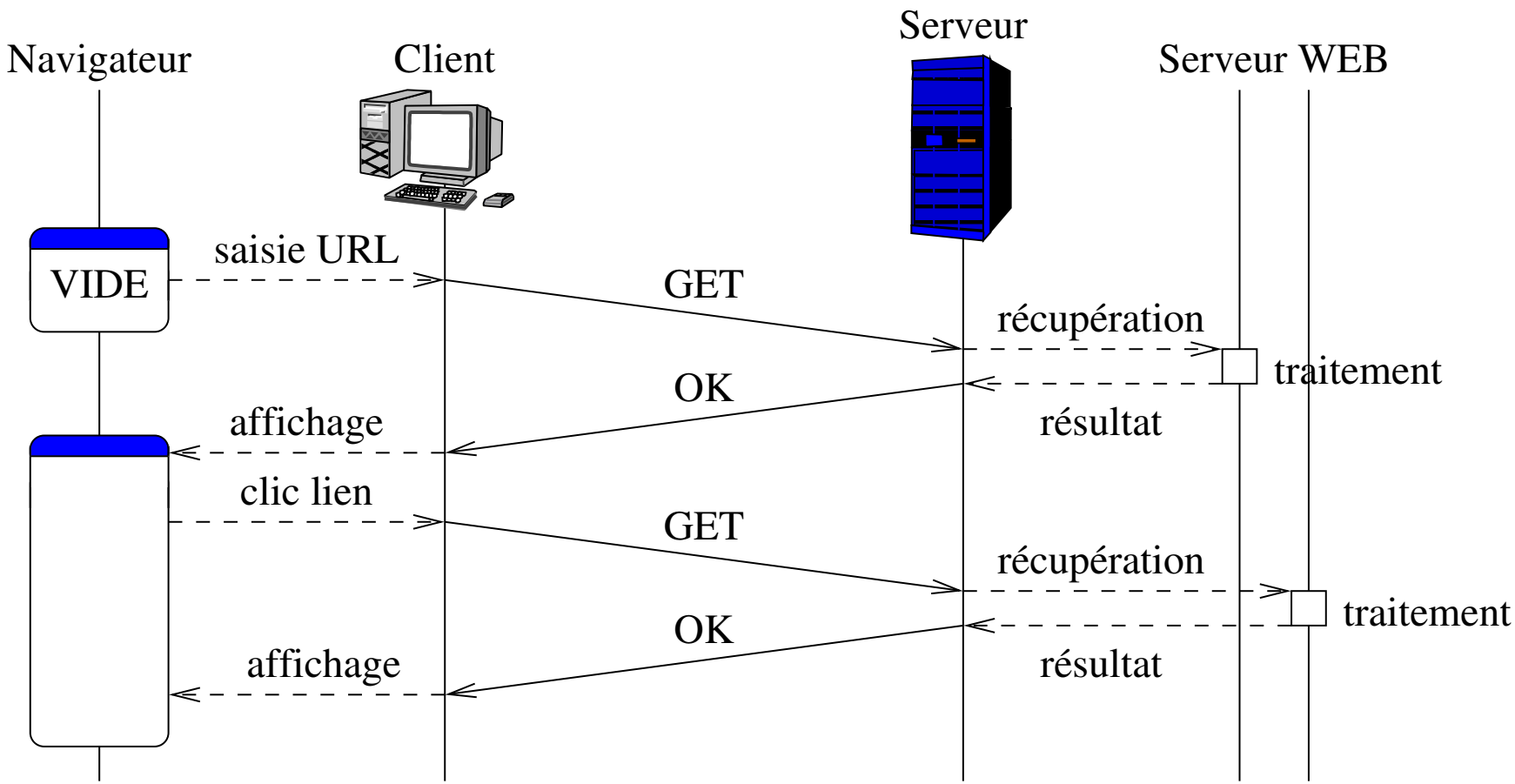
Utilisation d'un `cookie`

Différences d'appels



Utilisation d'une session

Différences d'appels



Utilisation d'AJaX (couplage possible avec les `cookies` et les sessions)

Présentation de la classe XMLHttpRequest

- Manipulation d'AJaX :
  ↪Utilisation de la classe `XMLHttpRequest`
- Possède un constructeur par défaut :
  ↪`var requeteHTTP = new XMLHttpRequest();`
- Permet l'envoi de requêtes HTTP et le traitement de la réponse
- Si plusieurs requêtes simultanées, plusieurs objets nécessaires
- Une fois créé, il faut initialiser l'objet :
  - Préparation de la requête
  - Modification de l'en-tête

Préparation de la requête et envoi

- Création de l'URL fourni à l'objet `XMLHttpRequest` :
  `↪requeteHTTP.open("GET", "toto.html");`
- Le prototype de la méthode `open` :
  `open(méthode, URL, synchrone, user, password)`
  - `méthode` : GET, POST, HEAD, PUT, DELETE
  - `URL` : URL
  - `synchrone` (option) : `true` si appel asynchrone (par défaut)
  - `user` et `password` (option) : authentification HTTP
- Une fois l'URL spécifié, envoi de la requête :
  `↪requeteHTTP.send();`

Appel synchrone

- Appel à `send` bloquant :
  ↪Instructions suivantes exécutées après la réponse reçue
  ↪L'utilisateur n'a pas la main pendant ce temps
- Spécifié lors de l'appel à la méthode `open` :
  ↪`requeteHTTP.open("GET", URL, false);`
- Troisième paramètre à `false`
  ↪Par défaut, c'est `true` (= appel asynchrone)

Appel asynchrone

- Appel à `send` non bloquant
- À la réception de la réponse : un *handler* (ou *callback*) est appelé
- Plusieurs types de *handler* suivant l'état de l'appel :
  - `onloadstart`, `onload`, `onloadend`
  - `onprogress`
  - `onabort`
  - `onerror`
  - `ontimeout`
- Pour spécifier le *handler* :
  ↪`requeteHTTP.onloadend = function() { ... };`
- Il s'agit d'un pointeur de fonction : possible d'utiliser une fonction existante
  ↪`requeteHTTP.onloadend = traitement;`
  ↪ La fonction `traitement` est définie ailleurs

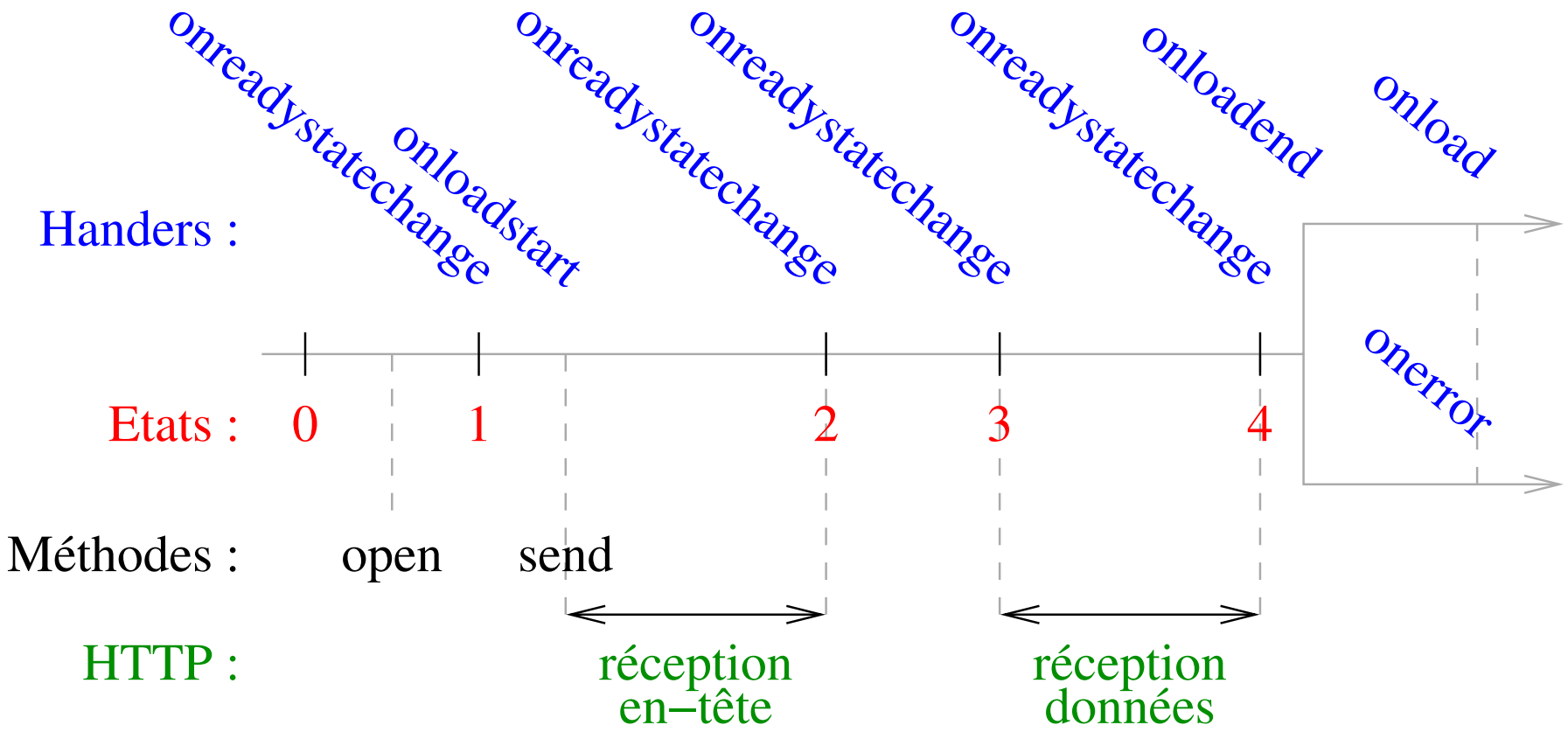
Handler générique

- Type de *handler* générique : `onreadystatechange`
  ↪Fonction associée appelée à chaque changement d'état de l'objet
- Le *handler* est appelé plusieurs fois ! ! !
- Comment savoir si la réponse a été reçue ?
  ↪Vérification de l'état de l'objet

L'état de l'objet

- L'objet `XMLHttpRequest` prend différents états successifs
- État courant récupéré à l'aide de l'attribut `readyState`
- Les différents états :
  - `UNSENT` (=0) : l'objet est construit
  - `OPENED` (=1) : la méthode `open` a été correctement invoquée
  - `HEADER_RECEIVED` (=2) : en-tête final reçu
  - `LOADING` (=3) : corps en cours de réception
  - `DONE` (=4) : toutes les données ont été reçues

Résumé d'un appel et états correspondants



Récupération du statut HTTP

- Deux attributs :
  - `status` : code d'état HTTP (exemple : 200)
  - `statusText` : message associé (exemple : OK)
- Utilisable dès que l'en-tête est reçu
- Par défaut à 0 (ou en cas d'erreur)

Récupération des champs de l'en-tête

- Méthode `getResponseHeader(nomDuChamp)`
- Récupère la valeur associée à un champ
- Par exemple :
  ↪`requeteHTTP.getResponseHeader("Content-type");`
- Récupération de tous les champs : `getAllResponseHeaders()`
  ↪Chaîne contenant tous les champs

Fixer un compte-à-rebours

- Possibilité de fixer un temps maximum pour la réponse
- Utilisation de l'attribut `timeout` (en millisecondes)
- Si la réponse n'est pas reçue avant l'expiration du compte-à-rebours :
  ↪*handler* `ontimeout` exécuté
- Autre solution pour interrompre une requête : méthode `abort()`

Attention

Attention au support des navigateurs...

Modifier l'en-tête

- Possibilité d'ajouter des champs dans l'en-tête HTTP :
  ↪À faire avant l'envoi de la requête
- Utilisation de la méthode `setRequestHeader` :
  ↪`setRequestHeader(nomDuChamp, valeur)`
- Si le champ existe, combinaison avec la valeur existante

Remarque sur les cookies

- La gestion des cookies est automatique en *Javascript*
  ↪`document.cookie`
- Pas de modification de l'en-tête manuelle nécessaire

Hello World - Description de l'exemple

- Page Web complétée dynamiquement :
  - Nouveaux éléments reçus via AJaX
  - Insertion dans le DOM
- Scripts utilisés :
  - Fichier HTML :
    - Contient le *Javascript* exploitant AJaX
    - Paragraphe vide complété avec la réponse AJaX
      ↪Possible aussi de le construire à l'exécution
  - Script PHP : génération de texte
- Deux possibilités proposées ici :
  - Version synchrone
  - Version asynchrone

Version synchrone - Fichier HTML

```
<html>
  <head>
    <script type="text/javascript">
      function exemple() {
        var requeteHTTP = new XMLHttpRequest();
        requeteHTTP.open("GET", "generateur.php", false);
        requeteHTTP.send();
        if((requeteHTTP.readyState == 4) &&
           (requeteHTTP.status == 200))
          document.getElementById("destAJaX").innerHTML =
            requeteHTTP.responseText;
      }
    </script>
  </head>
  <body>
    <h1>Exemple d'AJaX</h1>
    <p id="destAJaX"></p>
    <button type="button" onclick="exemple()"> Citation suivante </
        button>
  </body>
</html>
```

Version synchrone - Script PHP

```php
<?php
// Tableau contenant des chaînes de caractères
define("CITATIONS", [
                        "Citation␣1.␣<i>(Auteur␣1)</i>",
                        "Citation␣2.␣<i>(Auteur␣2)</i>",
                        "Citation␣3.␣<i>(Auteur␣3)</i>",
                        "Citation␣4.␣<i>(Auteur␣4)</i>",
                        "Citation␣5.␣<i>(Auteur␣5)</i>"
                    ]);

// Spécification du type de données envoyé (inutile)
header("Content-type:␣text/html");

// Écriture d'une case aléatoire
echo CITATIONS[rand(0, sizeof(CITATIONS) - 1)];
?>
```

Version synchrone - Exécution

1. Création du document contenant l'élément paragraphe vide
2. Lorsque le bouton est pressé, l'objet `XMLHttpRequest` est créé
3. La requête est ensuite créée puis envoyée
   ↪Attente de la réponse
4. Lorsque la réponse est reçue, l'exécution reprend :
   ↪Le texte est ajouté dans la balise
   ↪L'utilisateur reprend la main

Version asynchrone - onreadystatechange

```
<html>
  <head>
    <script type="text/javascript">
      function exemple() {
        var requeteHTTP = new XMLHttpRequest();
        requeteHTTP.onreadystatechange = function() {
          if((requeteHTTP.readyState == 4) &&
             (requeteHTTP.status == 200))
            document.getElementById("destAJaX").innerHTML =
              requeteHTTP.responseText;
        }
        requeteHTTP.open("GET", "generateur.php");
        requeteHTTP.send();
      }
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Version asynchrone - onload et onerror

```
<html>
  <head>
    <script type="text/javascript">
      var requeteHTTP = new XMLHttpRequest();
      requeteHTTP.onload = reussite;
      requeteHTTP.onerror = echec;

      function reussite() {
        document.getElementById("destAJaX").innerHTML = requeteHTTP.
            responseText;
      }
      function echec() {
        document.getElementById("destAJaX").innerHTML = "Erreur␣!";
      }
      function exemple() {
        requeteHTTP.open("GET", "generateur.php");
        requeteHTTP.send();
      }
    </script>
  </head>
  ...
```

Version asynchrone - Exécution

1. Création du document contenant l'élément paragraphe vide
2. Lorsque le bouton est pressé, l'objet `XMLHttpRequest` est créé
3. La requête est ensuite créée puis envoyée :
   ↪L'utilisateur reprend la main
4. Lorsque la réponse est reçue : exécution de la fonction *handler*
   ↪Le texte est ajouté dans la balise

Envoi de données - Description de l'exemple

- Reprise de l'exemple précédent :
  ↪Le client choisit maintenant la citation
- Solution retenue :
  ↪Envoi d'une valeur par la méthode GET
- Les données sont codées dans l'URL :
  ↪Ajout de "`?`" puis des couples séparés par "`&`"
  ↪Chaque couple : `nomVariable=valeur`
- Utilisation de la fonction *Javascript* `encodeURIComponent` :
  ↪Encodage pour que l'URL soit valide
  ↪Non obligatoire pour un entier

Fichier HTML

```
<html lang="fr">
  <head>
    <script type="text/javascript">
      function exemple(numero) {
        var requeteHTTP = new XMLHttpRequest();
        requeteHTTP.onreadystatechange = function() {
          if((requeteHTTP.readyState == 4) &&
             (requeteHTTP.status == 200))
            document.getElementById("destAJaX").innerHTML =
                requeteHTTP.responseText;
        }
        requeteHTTP.open("GET", "generateur.php?numero=" + numero);
        requeteHTTP.send();
      }
    </script>
  </head>
  <body>
    <h1> Citation du jour </h1>
    <p id="destAJaX"></p>
    <button type="button" onclick="exemple(0)"> Citation 1 </button>
    <button type="button" onclick="exemple(1)"> Citation 2 </button>
...
```

Script PHP

```php
<?php
// Tableau contenant des chaînes de caractères
define("CITATIONS", [
                        "Citation␣1.␣<i>(Auteur␣1)</i>",
                        "Citation␣2.␣<i>(Auteur␣2)</i>",
                        "Citation␣3.␣<i>(Auteur␣3)</i>",
                        "Citation␣4.␣<i>(Auteur␣4)</i>",
                        "Citation␣5.␣<i>(Auteur␣5)</i>"
                    ]);

// Spécification du type de données envoyé (inutile)
header("Content-type:␣text/html");

// Écriture de la case choisie (ou aléatoire si non spécifiée)
if(isset($_GET['numero']))
    $numero = intval($_GET['numero']) % sizeof(CITATIONS);
else
    $numero = rand(0, sizeof(CITATIONS) - 1);

echo CITATIONS[$numero];
?>
```

Envoi de données - Méthode POST

- Données non spécifiées dans l'URL
- Modifier l'en-tête avec le `Content-Type` suivant :
  ↪Pour les formulaires : `application/x-www-form-urlencoded`
- Données passées dans la méthode `send`
  ↪Attention à l'encodage !

```
function exemple(numero) {
  var requeteHTTP = new XMLHttpRequest();
  requeteHTTP.onreadystatechange = function() {
    if((requeteHTTP.readyState == 4) && (requeteHTTP.status == 200))
      document.getElementById("destAJaX").innerHTML =
          requeteHTTP.responseText;
  }
  requeteHTTP.open("POST", "generateur.php");
  requeteHTTP.setRequestHeader("Content-Type",
                               "application/x-www-form-urlencoded");
  requeteHTTP.send("numero=" + numero);
}
```

JQuery

- Bibliothèque *Javascript*
- Permet :
  - D'enrichir les possibilités de *Javascript*
  - Simplifier le développement
  - Système de sélecteurs puissant
  - Ajouter des effets
  - *etc.*
- À inclure comme un script *Javascript* :
  ↪Utiliser le CDN

AJaX avec *JQuery*

- Solution permettant de simplifier les appels AJaX
- Fonction `ajax` qui prend en paramètre :
  - Le type de requête HTTP (GET, POST, *etc.*)
  - L'URL
  - Les données
    ↪Format standard (`"nom=" + nom + "&prenom=" + prenom`)
    ↪JSON (`{'nom' : nom, 'prenom' : prenom }`)
  - Les *handlers* pour la réussite, l'échec
- Permet aussi simplement de modifier l'en-tête :
  - Type des données envoyées
    ↪Exemple : `application/x-www-form-urlencoded`
  - Type des données à recevoir
  - *etc.*
- Objet `jqXHR` retourné

Code générique d'une requête AJaX

```
requete = $.ajax(
  {
    type: 'POST',
    url: '',
    data: '',
    dataType: 'json'
  }
);
requete.done(
  function (reponse, texteStatut, jqXHR) {
    // Requête réussie
  }
);
requete.fail(
  function (jqXHR, texteStatut, erreurLevee){
    // Erreur !!!
  }
);
```

Exemple : les familles des séries télévisées

- Application permettant de récupérer les personnages de séries
- Constituée des fichiers suivants :
  - `index.html` : formulaire pour la saisie
  - `script.js` : script *Javascript* avec la requête AJaX
    ↪Interagit avec `donnees.php`
  - `series.json` : données brutes
  - `donnees.php` : retourne les données en fonction des saisies de l'utilisateur
- Fonctionnement :
  - L'utilisateur saisit le nom de la série et éventuellement, le nom de la famille
  - La liste des familles ou des membres est affichée

Extrait du fichier `index.html`

```
...
<label ref='serie'>Série</label>
<input type='text' name='serie' id='serie'/>
<label ref='famille'>Famille</label>
<input type='text' name='famille' id='famille'/>

<button type="button" id='bouton' onclick='javascript:recherche()'>
  Recherche
</button>

<p id="resultat"></p>

<script src="jquery.min.js"></script>
<script src="script.js"></script>
...
```

Le script PHP `donnees.php`

- Récupère le nom de la série et éventuellement, le nom de famille
  ↪En POST
- Chargement des données depuis le JSON
- Retourne un document JSON contenant :
  - Le code de retour (OK ou KO)
  - La liste des données (séries, familles, personnages)

Extrait du fichier *Javascript* `script.js` (1/3)

```
var requete = null;
function recherche() {
    requete = $.ajax({
        type: 'POST',
        url: 'donnees.php',
        data: { "serie" : $('#serie').val(), "famille" : $('#famille')
            .val() },
        dataType: 'json'
    });
    requete.done(function (response, textStatus, jqXHR) {
        if(response['code'] == 1) {
            if("familles" in response) {
                $('#resultat').text("").append("<ul>");
                for(i = 0; i < response['familles'].length; i++)
                    $('#resultat').append("<li>" +
                        response['familles'][i]['nom'] + "</li>");
                $('#resultat').append("</ul>");
            }
...
```

Extrait du fichier *Javascript* `script.js` (2/3)

```
            else {
                $('#resultat').text("").append("<ul>");
                for(i = 0; i < response['membres'].length; i++)
                    $('#resultat').append("<li>" +
                        response['membres'][i]['prénom'] + " " +
                        response['membres'][i]['nom'] + "</li>");
                $('#resultat').append("</ul>");
            }
        }
        else {
            $('#resultat').text(response['message']);
            if("donnees" in response) {
                $('#resultat').append("<ul>");
                for(i = 0; i < response['donnees'].length; i++)
                    $('#resultat').append("<li>"+response['donnees'][i
                        ]+"</li>");
                $('#resultat').append("</ul>");
            }
        }
    });
...
```

Extrait du fichier *Javascript* `script.js` (3/3)

```
...
    requete.fail(function (jqXHR, textStatus, errorThrown){
        $('#resultat').text("Problème de requête AJaX.");
        console.log(jqXHR);
    });
}
```

Le contexte

Nous souhaitons développer un éditeur de nouvelle. Chaque nouvelle est constituée d'un auteur, un message, et une date. Les données seront stockées dans une base de donnée à laquelle Laravel accèdera via un modèle. La structure de la BD est donnée par la migration ci-dessous.

```
public function up()
    {
        Schema::create('news', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('auteur');
            $table->string('message');
            $table->string('date');
            $table->timestamps();
        });
    }
```

Création des routes, vues et contrôleurs

```
php artisan make:model News
php artisan make:controller
     NewsController
```

```
// web.php

Route::view('/news', 'news');
Route::post('/news/post', '
    NewsController@store');
```

```
<!doctype html>
<html lang="fr">
<head>
    <title>Lecteur de News</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1
        ">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
        bootstrap/4.3.1/css/bootstrap.min.css">
</head>
<body>
    Lecteur de News
</body>
</html>
```

Création d'un formulaire

```
<body>
    <div class="container">
        <form id="myForm">
            <div class="form-group">
                <label for="nom">Nom de l'auteur :</label>
                <input type="text" class="form-control" id="nom"> </
                    div>
            <div class="form-group">
                <label for="message">Nouvelle :</label>
                <input type="text" class="form-control" id="message">
            </div>
            <div class="form-group">
                <label for="date">Date:</label>
                <input type="text" class="form-control" id="date">
            </div>
            <button class="btn␣btn-primary">Envoyer</button>
        </form>
    </div>
</body>
```

Importation de jQuery

Afin de simplifier la mise en place des échanges entre le client et le serveur, nous allons utiliser AJAX. Pour cela nous allons importer la librairie jQuery dans le fichier news.blade.php. Dans l'exemple suivant nous utilisons la version CDN de jQuery, mais vous pouvez aussi utiliser une version locale du fichier jQuery.

```
<script src="http://code.jquery.com/jquery-3.3.1.min.js"
                integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8
                    QtmkMRdAu8="
                crossorigin="anonymous">
</script>
```

Configurer une requête Ajax pour Laravel

Pour tout échange de données, Laravel exige l'utilisation du jeton CSRF habituellement défini dans une balise méta en insérant un champ `csrf_field()`.

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

Avec Ajax, lorsque l'on configure une requête, il faut en plus configurer l'en-tête de la requête afin qu'elle contienne le jeton CSRF.

L'intégration du token est réalisée par le code suivant :

```
$(document).ready(function(){
    $('#ajaxSubmit').click(function(e){
        e.preventDefault();
        $.ajaxSetup({
            headers: {'X-CSRF-TOKEN': $('meta[name="csrf-token"]').
                attr('content')}
        });
    });
});
```

Configurer une requête Ajax pour Laravel

On complète le script avec un nouvel appel à la fonction `jQuery.ajax ()` pour soumettre la demande au serveur avec les trois paramètres d'entrée.

```
.....
 $.ajax({
     url: "{{ url('/news/post') }}",
     method: 'post',
     data: {
         nom: $('#nom').val(),
         message: $('#message').val(),
         date: $('#date').val()
     },
     success: function(result){
         $('.alert ').show();
         $('.alert ').html(result.success);
     }});
```

Côté serveur

On ajoute dans le fichier NewsController.php le code suivant :

```
use App\News;

class NewsController extends Controller
{
    public function store(Request $request) {
        $news = new News();
        $news->auteur = $request->nom;
        $news->message = $request->message;
        $news->date = $request->date;
        $news->save();
        return response()->json([
            'success'=>'Vos données ont été correctement enregistrées'
        ]);

    }
}
```