
Appels de WS REST (HTTP) depuis js/ajax

Avec ajax , ça va briller!

Cadre des appels

Lorsqu'une requête http est initiée depuis du code javascript s'exécutant dans le contexte d'une page html , on dit que l'on effectue un appel "ajax" .

Le sigle Ajax correspondant à peu près à "asynchronous javascript activation framework" indique :

- le déclenchement non bloquant d'une requête http
- l'enregistrement d'une fonction "callback" qui sera automatiquement appelée en différé lorsque la réponse reviendra

NB : l'appel non bloquant peut être considéré comme asynchrone mais le protocole HTTP est un protocole de transport synchrone (avec timeout si la réponse ne revient pas dans un délai raisonnable).

Techniquement, un appel ajax s'effectue en s'appuyant sur un objet technique "XmlHttpRequest" fourni par tous les navigateurs pas trop anciens .

La partie Xml de XmlHttpRequest tient au fait qu'historiquement les premiers webServices normalisés (SOAP) étaient au format Xml. Bien que le terme "XmlHttpRequest" n'ait pas été changé pour des raisons de compatibilité ascendante du code javascript, il est possible de déclencher n'importe quelle requête HTTP depuis XHR , y compris des requêtes au format "JSON" .

Pour simplifier la syntaxe d'un appel ajax on peut éventuellement s'appuyer sur des bibliothèques "javascript" complémentaires ("jquery" , "fetch" , "RxJs" , ...) .

Le principe des appels "ajax" sert essentiellement à déclencher une requête HTTP suite à un événement utilisateur en vue d'obtenir des données permettant de réactualiser une partie de la page HTML courante . La page courante n'est pas entièrement remplacée par une autre. Seule une sous partie de celle ci est ajustée par code js/DOM (Document Object Model).

Certaines applications , dites "SPA: Single Page Application" sont entièrement bâties sur ce principe . Au lieu de switcher de pages html on switch de sous pages (<div ...>....</div>) .

XHR (XmlHttpRequest) dans tout navigateur récent

Exemple :

```
function makeAjaxRequest(callback) {  
    var xhr = new XMLHttpRequest();  
  
    xhr.onreadystatechange = function() {  
        if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {  
            callback(xhr.responseText);  
        }  
    };  
  
    xhr.open("GET", "handlingData.php", true);  
    xhr.send(null);  
}  
  
function readData(sData) {  
    if (sData!=null) {  
        alert("good response");  
        // + display data with DOM  
    } else {  
        alert("empty response");  
    }  
}  
  
makeAjaxRequest(readData);
```

variations en mode "POST" :

```
xhr.open("POST", "handlingData.php", true);  
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  
xhr.send("param1=xx&param2=yy");
```

```
xhr.open("POST", "json-handler");  
xhr.setRequestHeader("Content-Type", "application/json");  
xhr.send(JSON.stringify({prenom:"Jean", nom:"Bon"}));
```

Appel ajax via jQuery

La syntaxe des appels "ajax" est un peu plus explicite en s'appuyant sur la bibliothèque "jquery".

Exemple :

```
<html>
<head>
  <meta charset="ISO-8859-1">
  <title>browse-spectacles</title>
  <script src="lib/jquery-3.3.1.min.js"></script>
  <script src="js/my-jq-ajax-util.js"></script>
</script>
$(function() {

  //appel ajax pour récupérer la liste des catégories et remplir le <select>
  $.ajax({
    type: "GET",
    url: "spectacle-api/public/spectacle/allCategories",
    contentType : "application/json",
    success: function (data,status,xhr) {
      if (data) {
        var categoryList = data;
        for(categoryIndex in categoryList){
          var category=categoryList[categoryIndex];
          $('#selectCategory').append('<option value="'+ category.id +'"'+
            category.id + ' (' + category.title + ')</option>');
        }
        //$("#spanMsg").html(JSON.stringify(data));
      }
    },
    error: function( jqXHR, textStatus, errorThrown ){
      $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
  }); //end $.ajax
});

</script>
</head>
<body>
  <h3> BROWSE Spectacles </h3>
  categorie : <select id="selectCategory"> </select><br/>
  ... <span id="spanMsg"></span> <br/>...
</body>
</html>
```

fonctions utilitaires dans *js/my-jq-ajax-util.js*

```
function setSecurityTokenForAjax(){

  var authToken = sessionStorage.getItem("authToken");
  //localStorage.getItem("authToken");

  $(document).ajaxSend(function(e, xhr, options) {
```

```

        //retransmission du jeton d'authentification dans l'entête http de la requete ajax
        xhr.setRequestHeader('Authorization','Bearer '+ authToken);
    });
}

function xhrStatusToErrorMessage(jqXHR){
    var errMsg = "ajax error";//by default
    var detailsMsg="";//by default
    console.log("jqXHR.status="+jqXHR.status);
    switch(jqXHR.status){
        case 400 :
            errMsg = "Server understood the request, but request content was invalid.";
            if(jqXHR.responseText!=null)
                detailsMsg = jqXHR.responseText;
            break;
        case 401 :
            errMsg = "Unauthorized access (401)"; break;
        case 403 :
            errMsg = "Forbidden resource can't be accessed (403)"; break;
        case 404 :
            errMsg = "resource not found (404)"; break;
        case 500 :
            errMsg = "Internal server error (500)"; break;
        case 503 :
            errMsg = "Service unavailable (503)"; break;
    }
    return errMsg+" "+detailsMsg;
}

```

Variation en mode "POST" et "[application/x-www-form-urlencoded](#)"

```

var dataJsObject = { prenom : "jean", nom : "Bon", taille: 175 } ;
$.ajax({
    type: "POST",
    url: "./my-api/person",
    contentType : "application/x-www-form-urlencoded; charset=utf-8",
    data: $.param(dataJsObject),
    dataType : 'json_or_text_or_...',
    beforeSend: function (xhr) {
        xhr.setRequestHeader ("Authorization",
                                "Basic " + btoa("usernameXy" + ":" + "passwordXy"));
    },
    success: ... , error: ....
}); //end $.ajax

```

Variation en mode "POST" et "JSON" in/out :

```
//setSecurityTokenForAjax();//js/my-jq-ajax-util.js
$.ajax({
    type: "POST",
    url: "spectacle-api/spectacle",
    data: JSON.stringify(spectacleAdditionJsObject),
    dataType: "json",
    contentType: "application/json",
    success: function (data,status,xhr) {
        if (data) {
            $("#spanMsg").html(JSON.stringify(data));
        }
    },
    error: function( jqXHR, textStatus, errorThrown ){
        $("#spanMsg").html( xhrStatusToErrorMessage(jqXHR) );
    }
});//end $.ajax
```

Api fetch

Api récente (syntaxe concise basé sur enchaînement asynchrone et "**Promise**") mais pas encore supporté par tous les navigateurs.

Exemple :

```
fetch('./api/some.json')
    .then(
        function(response) {
            if (response.status !== 200) {
                console.log('Problem. Status Code: ' + response.status);
                return;
            }
            // Examine the text in the response :
            response.json().then(function(data) {
                console.log(data);
            });
        }
    )
    .catch(function(err) {
        console.log('Fetch Error :-S', err);
    });
```

Appel ajax via RxJs (api réactive)

Le framework "RxJs" lié au concept de "programmation asynchrone réactive" est assez sophistiqué et permet de déclencher une série de traitements d'une façon assez indépendante de la source de données (ex : données statiques , réponse ajax , push web-socket , ...) .

RxJs peut soit être directement utilisé en tant que bibliothèque javascript dans une page HTML , soit être utilisé via "typescript et le framework Angular 2,4,5,6 ou autre) .

Attention à la version utilisée (différences significatives dans la version récente de RxJs accompagnant Angular 6) .

--> une bonne présentation de RxJS est accessible au bout de l'URL suivante

<https://www.julienpradet.fr/tutoriels/introduction-a-rxjs/>

--> exemples d'appel ajax via RxJs et de config proxy dans le support de cours "Angular 4,5,6"