Appeler un service web en JavaScript

Michaël Mathieu

29 décembre 2023

Résumé

L'appel à un service web en JavaScript introduit plusieurs notions intéressantes : authentification auprès d'un service web, transmission de paramètres, exploitation du résultat au format `json`.

Table des matières

1 Préambule

Il existe plusieurs manières pour faire l'appel à un service web en JavaScript, notamment `XMLHttpRequest`, `ajax`, et `Fetch`.

Le premier mécanisme qui fut disponible était `XMLHttpRequest`.
L'utilisation d'`ajax` (fonction fournie par la bibliothèque *jQuery*) permet une utilisation simplifiée d'`XMLHttpRequest`, mais nécessite la dépendance à *jQuery* (ce qui est déjà le cas avec l'utilisation de *Bootstrap*). Depuis 2017 (apparu en 2015 sous forme expérimentale), l'API `Fetch` est une alternative à `XMLHttpRequest` qui est disponible en standard sur les navigateurs modernes.

A l'heure où nous écrivons ces lignes, `XMLHttpRequest` est toujours utilisé par la bibliothèque *jQuery*, est maintenu et n'est pas déprécié. Il nous paraît donc toujours pertinent de garder dans ce document ce mécanisme.
Ainsi, chaque exemple sera présenté avec `XMLHttpRequest`, `ajax`, et `Fetch`.
En termes de performances, ces trois mécanismes sont identiques.

La seule exigence, qui nous semble raisonnable, est de choisir un mécanisme pour un projet et de toujours utiliser le même en évitant le panachage.

Pour les exemples de ce document, nous avons une simple page HTML 5 avec deux boutons et une zone vide :

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
      <title>Call WS</title>
   </head>
   <body>
      <button onclick="callWS()">Call WS</button>
      <div id="resultWS"></div>

      <script type="text/javascript">
         function callWS() {
             ...
         }
      </script>
   </body>
</html>
```

2 Appel à un service non–sécurisé

Ici, nous allons utiliser un service web qui liste les pays : `https://restcountries.com`.

2.1 XMLHttpRequest

```
function callWS() {
  const URL = "https://restcountries.com/v3.1/all";
  let xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (this.readyState == XMLHttpRequest.DONE) {
      // https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState
      if (this.status === 200) {
        // http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml
        let data = JSON.parse(this.responseText);
        // use data
      } else {
        console.error(this.status + " when call " + URL);
        alert(this.status);
      }
    }
  };
  xhr.open("GET", URL, true);
  xhr.send();
}
```

2.2 ajax

```
function callWS() {
  $.ajax({
    type: "GET",
    url: "https://restcountries.com/v3.1/all",
    success: function(data) {
      // use data
    },
    error: function(jqXHR, textStatus, errorThrown) {
      alert("Error " + textStatus);
    }
  });
}
```

2.3 Fetch

```
function callWS() {
  fetch("https://restcountries.com/v3.1/all", {
      method: "GET"
    })
    .then(function(response) {
      if (response.status !== 200) {
        alert('Error ' + response.status);
        return;
      }

      response.json().then(function(data) {
        // use data
      });
    })
    .catch(function(e) {
      alert('Error ' + e);
    });
}
```

3 Appel à un service sécurisé

Nous allons prendre comme exemple le service web `openexchangerates.org`, un service qui fournit le taux de change actuel pour la plupart des monnaies.
Ce service web présente l'avantage d'avoir un plan gratuit, et de nécessiter une authentification par header notamment ; cela nous permettra de voir comment ajouter un header personnalisé à un appel.

Pour commencer, il faut obtenir votre clé : `https://openexchangerates.org/signup/free`
Dans l'exemple ci–dessous, nous imaginons que notre clé est *API_KEY*.

La clé doit être ajoutée dans le header de chacune de nos requêtes comme cela est indiqué ici `https://docs.openexchangerates.org/reference/authentication`

3.1 XMLHttpRequest

```
const URL = "https://openexchangerates.org/api/latest.json";
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (this.readyState == XMLHttpRequest.DONE) {
    if (this.status == 200) {
      let data = JSON.parse(this.responseText);
      // use data
    } else {
      console.error(this.status + " when call " + URL);
      alert(this.status);
    }
  }
};
xhr.open("GET", URL, true);
xhr.setRequestHeader("Authorization", "Token API_KEY");
xhr.send();
```

3.2 ajax

```
$.ajax({
  type: "GET",
  headers: {
    "Authorization": "Token API_KEY"
  },
  url: "https://openexchangerates.org/api/latest.json",
  success: function(data) {
    // use data
  },
  error: function(jqXHR, textStatus, errorThrown) {
    alert("Error " + textStatus);
  }
});
```

3.3 Fetch

```
fetch("https://openexchangerates.org/api/latest.json", {
    method: "GET",
    headers: {
      "Authorization": "Token API_KEY"
    }
  })
  .then(function(response) {
    if (response.status !== 200) {
      alert('Error ' + response.status);
      return;
    }

    response.json().then(function(data) {
      // use data
    });
  })
  .catch(function(e) {
    alert('Error: ' + e);
  });
```

3.4 Authentification Basic

Cette manière de s'authentifier consiste à envoyer à chaque requête les credentials au serveur. Le nom d'utilisateur et le mot de passe (séparés par :) doivent être encodés en Base64 (ASCII) [1], mais ne sont pas cryptés directement. Pour qu'ils le soient, il suffit que la communication soit en HTTPS.

Exemple

Avec le nom d'utilisateur *jc@dusse.com* et le mot de passe *EtoileDesNeiges* :

Credentials à encoder en Base64 (ASCII)	Donnée à placer dans le header
`jc@dusse.com:EtoileDesNeiges`	`amNAZHVzc2UuY29tOkV0b2lsZURlc05laWdlcw==`

1. Javascript fournit la fonction `btoa` pour convertir une chaîne de caractères en Base 64 ; la fonction `atob` fait l'inverse.

4 Utilisation des données au format json

Si au début des services web, le standard était *XML*, désormais, afin d'optimiser le temps de transfère des données, le format *json* est la norme.

4.1 Utiliser un object JavaScript

Nous pouvons rencontrer deux types de réponses :

1. un objet qui contient d'autres objets dont certains sont des tableaux
2. un tableau

4.1.1 Objet

On identifie un objet au fait qu'il commence par une accolade ouvrante {
Prenons l'exemple d'un pays :

```
{
      "name":{
          "common":"Switzerland",
          "official":"Swiss Confederation"
      },
      "cca2":"CH",
      "capital":"Bern",
      "region":"Europe",
      "borders":[
         "AUT",
         "FRA",
         "ITA",
         "LIE",
         "DEU"
      ]
}
```

Hormis l'attribut *borders*, tous les autres attributs sont des chaînes de caractères. Nous pouvons donc les utiliser de cette manière dans le code :

```
success: function (data) {
  let country = data;
  alert("Nous traitons le pays " + country.name.common +
        ". La capitale est " + country.capital);
},
```

4.1.2 Tableau

Dans l'exemple ci–dessus, l'attribut *borders* est un tableau.
On identifie un tableau au fait qu'il commence par un crochet ouvrant [

Le service web `restcountries.com` retourne directement un tableau contenant les pays.

```
[ ...,
   {
      "name":{
          "common":"Switzerland",
          "official":"Swiss Confederation"
      },
      ...,
      "borders":[
         "AUT",
         "FRA",
         "ITA",
         "LIE",
         "DEU"
      ]
    },
   ...
]
```

5 Exemple d'utilisation simple des données

Voici un exemple pour exploiter les données retournées par le service :

```
function callWS() {
  fetch("https://restcountries.com/v3.1/all", {
      method: "GET"
    })
    .then(
      function(response) {
        if (response.status !== 200) {
          console.log('Error, status code: ' + response.status);
          return;
        }

        response.json().then(function(data) {
          let result = "";
          for (let i = 0; i < data.length; i++) {
            let country = data[i];
            result += '<tr><td><img src="' + country.flags.svg +
              '" style="width:100px;"></td><td>' + country.name.common + '</td><td>' +
              country.cca2 + '</td><td>' + country.capital + '</td></tr>';
          }
          resultWS.innerHTML = '<table  border="1"><tr><th></th><th>Nom du pays</th>' +
            '<th>Code ISO 3166-1 alpha-2</th><th>Capitale</th></tr>' + result + '</table>';
        });
      }
    )
    .catch(function(err) {
      console.error('Fetch Error: ' + err);
    });
}
```

Résultat :

	Nom du pays	Code ISO 3166-1 alpha-2	Capitale
	Iceland	IS	Reykjavik
	Japan	JP	Tokyo
	New Caledonia	NC	Nouméa
	Somalia	SO	Mogadishu
			

6 Envoyer des données – Poster un formulaire

Pour les exemples ci–dessous, nous partons du principe que la constant `URL` contient l'URL du endpoint de l'API REST où les données sont envoyées.

6.1 Données brutes

Utilisé la plupart du temps pour envoyer un JSON (`Content-Type: application/json`), mais peut aussi envoyer un formulaire (`Content-Type: application/x-www-form-urlencoded`).

Ainsi, le payload ressemblera à :

```
// JSON
{"firstName":"Jean-Claude","lastName":"Dusse"}

// x-www-form-urlencoded
firstName=Jean-Claude&lastName=Dusse
```

6.1.1 XMLHttpRequest

```
let xhr = new XMLHttpRequest();
xhr.open("POST", URL, true);
xhr.onreadystatechange = function() {
  if (this.readyState == XMLHttpRequest.DONE && this.status == 200) {
    // use this.responseText
  }
};

// JSON
xhr.setRequestHeader("Content-Type", "application/json");
xhr.send(JSON.stringify({
  firstName: "Jean-Claude",
  lastName: "Dusse"
}));

// x-www-form-urlencoded
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
let body = new URLSearchParams();
body.append('firstName', 'Jean-Claude');
body.append('lastName', 'Dusse');
xhr.send(body.toString());
```

6.1.2 ajax

```
// JSON
$.ajax({
  type: "POST",
  contentType: "application/json",
  data: JSON.stringify({
    firstName: "Jean-Claude",
    lastName: "Dusse"
  }),
  url: URL,
  success: function(data) {
    // use data
  },
  error: function(jqXHR, textStatus, errorThrown) {
    alert("Error " + textStatus);
  }
});
```

```
// x-www-form-urlencoded
$.ajax({
  type: "POST",
  data: {
    firstName: "Jean-Claude",
    lastName: "Dusse"
  },
  url: URL,
  success: function(data) {
    // use data
  },
  error: function(jqXHR, textStatus, errorThrown) {
    alert("Error " + textStatus);
  }
});
```

6.1.3 Fetch

```
// JSON
fetch(URL, {
    method: "POST",
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      firstName: "Jean-Claude",
      lastName: "Dusse"
    })
  })
  .then(function(response) {
    if (response.status !== 200) {
      alert('Error ' + response.status);
      return;
    }

    response.json().then(function(data) {
      // use data
    });
  })
  .catch(function(e) {
    alert('Error: ' + e);
  });
```

```
// x-www-form-urlencoded
let body = new URLSearchParams();
body.append('firstName', 'Jean-Claude');
body.append('lastName', 'Dusse');

fetch(URL, {
    method: "POST",
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: body.toString()
  })
  .then(function(response) {
    if (response.status !== 200) {
      alert('Error ' + response.status);
      return;
    }

    response.json().then(function(data) {
      // use data
    });
  })
  .catch(function(e) {
    alert('Error: ' + e);
  });
```

6.2 FormData

L'intérêt d'utiliser le `FormData` est essentiellement pour permettre d'envoyer plusieurs types de données. Par exemple, du texte et un fichier (blob) [2]. Le `Content-Type` est `multipart/form-data`.

6.2.1 XMLHttpRequest

```
const formData = new FormData();
formData.append("firstName", "Jean-Claude");
formData.append("lastName", "Dusse");

let xhr = new XMLHttpRequest();
xhr.open("POST", URL, true);
xhr.onreadystatechange = function() {
  if (this.readyState == XMLHttpRequest.DONE && this.status == 200) {
    resultWS.innerHTML = this.responseText;
  }
};
xhr.send(formData);
```

2. https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects

6.2.2 ajax

```
const formData = new FormData();
formData.append("firstName", "Jean-Claude");
formData.append("lastName", "Dusse");

$.ajax({
  type: "POST",
  processData: false,
  contentType: false,
  data: formData,
  url: URL,
  success: function(data) {
    // use data
  },
  error: function(jqXHR, textStatus, errorThrown) {
    alert("Error " + textStatus);
  }
});
```

6.2.3 Fetch

```
const formData = new FormData();
formData.append("firstName", "Jean-Claude");
formData.append("lastName", "Dusse");

fetch(URL, {
    method: "POST",
    body: formData
  })
  .then(function(response) {
    if (response.status !== 200) {
      alert('Error ' + response.status);
      return;
    }

    response.json().then(function(data) {
      // use data
    });
  })
  .catch(function(e) {
    alert('Error: ' + e);
  });
```