



XMLHttpRequest

```
cting=false;e.selected=true;e.startselected=true;c.
 end(a.ui.selectable, {version: "1.8.16"})})(jQuery);
    idget("ui.sortable",a.ui.mouse,{widgetEventPrefix:
 ent", axis:false, connectWith:false, containment:false,
lse,placeholder:false,revert:false,scroll:true,
        this.containerCache={};this.elem
  this.floating=this.items.length?d.axi
 *.items[0].item.css("display")):false;
 (ed").removeData("sortable").unbind('
this},_setOption:func ≥0 (d,c){if(d==
his.options[d]=c;this.widget()[c?"addCl
otype._setOption.apply(this,arguments))
.options.disabled||this.options.typess
fs, "sortable-item") == h) {e=a(this); retu
(se;a(this.options.handle,e).
rrentItem=e; this._removeCurrentsFrom
 anecontainer=this:this.refreshPosi
                     a loftl:this.helpe
```

Signature des méthodes

.open(method, url, async, user, pass)

- -> method: "GET" / "POST"
- -> url : URL de la requête
- -> async : requête asynchrone/synchrone
- -> user : username (HTTP BASIC)
- -> pass : password (HTTP BASIC)

.send(data)

-> date : données à envoyé au serveur

.setRequestHeader(header, value)

- -> header : nom du header à modifier
- -> value : value du header

Utilisation du XMLHttpRequest

```
// Initialisation de l'objet
var request = new XMLHttpRequest();

// Spécification de la requête (synchrone)
request.open("GET", "test.php", false);
// Spécification de la requête (asynchrone)
request.open("GET", "test.php", true);

// Envoi GET
request.send();
// Envoi POST
request.send(data);
```

```
// Récupération du résultat (synchrone)
request.send();
request.getResponseText();
request.getStatus();

// Récupération du résultat (asynchrone)
request.onload = function(event) {
  console.log(this.status); // retourne le status code
  console.log(this.responseText); // retourne le
  body
}
request.send();
```

```
cting=false;e.selected=true;e.startselected=true;c.
 emd(a.wi.selectable, {version: "1.8.16"})})(jQuery);
    dget("ui.sortable",a.ui.mouse,{widgetEventPo
     .axis:false,connectWith:false,containment:false
    placeholder:false,revert:false,scroll:true,
         this.containerCache={};this.elem
   his.floating=this.items.length?d.axi
    tems[0].item.css("display")):false;
  ed").removeData("sortable").unbind("
this},_setOption:func €0 (d,c){if(d==.
his.options[d]=c;this.widget()[c?"addC
otype._setOption.apply(this,arguments)
.options.disabled||this.options.typess
18, "sortable-item") == h) {e=a(this); retu
(se;a(this.options.handle,e).
rentItem=e; this._removeCurrentsFrom
                      - leftl:this.help
```

Utilisation de FormData

 Objet permettant de gérer facilement les données POST à partir d'un formulaire

// Initialisation

request.send(data);



Destructuration d'objets

Utilisation de la déstructuration/décomposition

// Principe de base

var [a, b, c, d, e = 5] = [1, 2, 3, 4]; console.log(a, b, c, d, e); // 1 2 3 4 5

// Filtrer les valeurs

var [a, b] = [1, 2, 3, 4]; console.log(a, b); // 1 2

var [a, , c] = [1, 2, 3, 4]; console.log(a, c); // 13

// Utilisation de ... (spread/rest operator)

var [a, b, ...c] = [1, 2, 3, 4]; console.log(a, b, c); // 1 2 [3, 4]

Le spread operator doit toujours être en dernier

// Changement de nom des variables

var {a: aa, b] = {a: 1, b: 2, 3, 4}; console.log(aa, b); // 12

// Décomposition imbriquée

var {a: aa, b: {b1: bb}} = {a: 1, b: {b1: 21, b2:22}}; console.log(aa, bb); // 1 21 Seul les valeurs les plus imbriquées sont conservées

// Equivalent JS pure => la décomposition génère de nouveaux objets

var right = {a: 1, b: 2, 3, 4};
var {a: aa, b] = right;
==> var aa = Object.assign({}, right.a);
var b = Object.assign({}, right.b);