Des modèles clé-en-main pour

Organiser son code JavaScript

Qui suis-je ?

Thomas ZILLIOX, développeur web freelance sur Lyon.

- Spécialisé dans **l'industrialisation du CSS** :
  Formation, conseil, mise en place d'outils et de bonnes pratiques.
- Développe aussi en JS & PHP ;
- Blog (*rarement*) sur mon site **tzi.fr** ;
- Tweete (*plus souvent*) sur **@iamtzi**.

Revoir cette présentation en ligne **git.io/ModulesJS**.

1. Pourquoi organiser

1.A. Pourquoi organiser son code en modules ?

Le but principal est de limiter l'exposition inutile de variables et de fonctions pour :

1. **Éviter les conflits** (*et les bugs*) ;
2. Améliorer la maintenabilité ;
3. Permettre la réutilisation de projet en projets ;
4. Expliciter les options.

1.B. D'où viennent les conflits ?

Savez-vous ce que va afficher ce script ?

```
function id() {
    key = '';
    for (i = 0; i < 16; i++) {
        key += Math.floor(Math.random() * 10);
    }
    return key;
}

var id = id();
console.log(id);

var id2 = id();
console.log(id2);
```

1.B. D'où viennent les conflits ?

L'espace de nommage est partagé entre variables et fonctions.

```
// A function
function foo() {
}

console.log(typeof foo);

// A string, with a conflicted name
var foo = 'BrownBagLunch';

console.log(typeof foo);
```

1.B. D'où viennent les conflits ?

Savez-vous ce que va afficher ce script ?

```
for (var i = 0; i < 10; i++) {
    console.log(getRandomKey(16));
}

function getRandomKey(length) {
    key = '';
    for (i = 0; i < length; i++) {
        key += Math.floor(Math.random() * 10);
    }
    return key;
}
```

1.B. D'où viennent les conflits ?

L'espace de nommage est partagé en cascade.

```
var organizer = 'BrownBagLunch';
init();
console.log('event', event);

// Create a new `cascading` scope
function init() {
    var event = 'Atelier sur JavaScript';
    console.log('organizer', organizer);
}
```

1.B. D'où viennent les conflits ?

Savez-vous ce que va afficher ce script ?

```
function newButton() {
    return {};
}

var buttonList = [];
for (var i = 0; i < 5; i++) {
    var button = newButton();
    button.onclick = function(){
        console.log(i);
    };
    buttonList[i] = button;
}

for (var j = 0; j < buttonList.length; j++) {
    buttonList[j].onclick();
}
```

1.B. D'où viennent les conflits ?

L'espace de nommage est lié, il n'est pas figé à la déclaration.

```
var organizer = 'Google';
var closure1 = function() {
    console.log(organizer);
};

organizer = 'BrownBagLunch';
var closure2 = function() {
    console.log(organizer);
};

closure1();
closure2();
```

1.B. D'où viennent les conflits ?

L'espace de nommage est celui de déclaration, pas celui d'exécution.

```
// Returns a closure,
// a function that can be used outside its definition scope
function getClosure() {
    var organizer = 'BrownBagLunch';
    return function() {
        console.log(organizer);
    };
}

var organizer = 'Google';
var closure = getClosure();
closure();
```

1.B. D'où viennent les conflits ?

L'éxecution d'une fonction crée un nouvel espace de nommage.

```
// Returns a closure,
// a function that can be used outside its definition scope
function getClosure(organizer) {
    return function() {
        console.log(organizer);
    };
}

var closure1 = getClosure('Google');
var closure2 = getClosure('BrownBagLunch');

closure1();
closure2();
```

2. Écrire des modules

2.A. L'état de l'art des modules JavaScript

L'histoire des modules JavaScript en un slide :

1. Pas de modules, un seul fichier “main.js” ;
2. **Modules en augmentant l'isolation**, 1 module = 1 fichier ;
3. Modules en utilisant des patterns CommonJS, AMD, UMD ;
4. Modules natifs node ;
5. Modules natifs ES6 / ES2015.

2.B. Isoler son code

Les 50 nuances de fonctions.

```
console.log(typeof foo1);

// Declared function: Interpreted on compilation time
function foo1() {
}

// Anonymous assigned function: Interpreted only on Runtime
var foo2 = function () {
};

// Named assigned function: Have a name in stack traces
var foo3 = function foo3F() {
};
```

2.B. Isoler son code

Tous les types natifs JavaScript sont des objets. C'est plus sûr de faire du "casting".

```
console.log( (true).toString() );
console.log( (418).toString() );
console.log( ('BrownBagLunch').toString() );
console.log( (/Lunch$/).test('BrownBagLunch') );
console.log( ([]).toString() );
console.log( ({}).toString() );
console.log( (function Module(){}).name );
```

2.B. Isoler son code

Utiliser les fonctions anonymes pour protéger ses variables.

```
// Auto-executed anonymous function
var anonymous = function anonymousF() {
    var organizer = 'BrownBagLunch';
    // [...] my module here
};
anonymous();

console.log(typeof anonymous);
console.log(typeof organizer);
```

2.B. Isoler son code

Utiliser des fonctions en mode strict.

```
(function () {
    'use strict';
    // Access global variables
    foo = "bar";
    // Failing assignements
    NaN = 5;
    // Delete undeletable properties
    delete Object.prototype;
    // Double parameter name
    function sum(x, x) {}
    // ...
})();
```

2.C. Module Vanilla I - Avec isolation

Un modèle pour exposer un service global.

```
// Definition - File `myModule.js`
var myModule = (function myConstructor() {
    "use strict";

    // [...] The module code

    // Return the main service function
    return myService;
})();

// Usage - File `main.js`
// [...]
myModule(option1, option2);
```

2.C. Module Vanilla I - Avec isolation

Un exemple avec un service de sortie d'écran.

```
// Definition - File `output.js`
var outputModule = (function outputConstructor() {
    "use strict";

    // [...] The module code
    function output(str) {
        console.log(str);
    }

    // Return the main service function
    return output;
})();

// Usage - File `main.js`
// [...]
outputModule('BrownBagLunch');
```

2.D. Module Vanilla II - Avec multi-services

Un modèle pour exposer un ensemble de services.

```
// Definition - File `myModule.js`
var myModule = (function myConstructor() {
    "use strict";

    // [...] The module code
    function myService1() {}

    function myService2() {}

    // Return a set of function
    return {
        myService1: myService1,
        myService2: myService2
        // [...]
    };
})();

// Usage - File `main.js`
// [...]
myModule.myService1(option1, option2);
```

2.D. Module Vanilla II - Avec multi-services

Un exemple avec un service de sortie d'écran.

```
// Definition -File `output.js`
var outputModule = (function outputConstructor() {
    "use strict";

    // [...] The module code
    var buffer = [];

    function push(value) {
        buffer.push(value);
        return this;
    }

    function output() {
        console.log(buffer.join(' '));
        buffer = [];
    }

    // Return a set of function
    return {
        push: push,
        output: output
    };
})();

// Usage - File `main.js`
// [...]
outputModule.push('Brown').push('Bag').push('Lunch').output();

```

2.D. Module Vanilla II - Avec multi-services

Un exemple avec un service de géolocalisation (*simplifié*).

```
// Definition - File `geolocation.js`
var geolocationModule = (function geolocationConstructor() {
    "use strict";

    // Protected scope
    var timeout = 10000;
    var maximumAge = 300000;

    function isAvailable() {
        return 'geolocation' in navigator;
    }

    function setTimeout(value) {
        timeout = value;
    }

    function setMaximumAge(value) {
        maximumAge = value;
    }

    function geolocate(successCallback, errorCallback) {
        if (!isAvailable()) {
            errorCallback();
            return;
        }
        navigator.geolocation.getCurrentPosition(successCallback, errorCallback, {
            timeout: timeout,
            maximumAge: maximumAge
        });
    }

    // Exposed function
    return {
```

2.E. Module Vanilla III - Avec multi-instances

Un modèle pour gérer plusieurs instances.

```
// Definition - File `myModule.js`
function myConstructor() {
    "use strict";

    // [...] The module code
    function myService1() {}

    function myService2() {}

    // Return a set of function
    return {
        myService1: myService1,
        myService2: myService2
        // [...]
    };
}

// Usage - File `main.js` or in another module !
// [...]
var myModule = myConstructor();
myModule.myService1(option1, option2);
```

2.E. Module Vanilla III - Avec multi-instances

Un exemple avec un service de sortie d'écran.

```
// Definition - File `output.js`
function outputConstructor() {
    "use strict";

    // [...] The module code
    var buffer = [];

    function push(value) {
        buffer.push(value);
        return this;
    }

    function output() {
        console.log(buffer.join(' '));
        buffer = [];
    }

    // Expose the service in the ask set
    return {
        push: push,
        output: output
    };
};

// Usage - File `main.js` or in another module !
// [...]
var outputModule = outputConstructor();
outputModule.push('La').push('Brown').push('Bag').push('Lunch').output();
// or
var modules = {};
modules.outputModule = outputConstructor();
modules.outputModule.push('Isolated').push('JavaScript').push('FTW').output();
```

2.F. Module Vanilla IV - Avec dépendances dynamiques

Un modèle pour gérer les dépendances.

```
// Definition - File `myModule.js`
function myConstructor(dependency1, dependency2) {
    "use strict";

    // [...] The module code
    function myService1() {}

    function myService2() {}

    // Return a set of function
    return {
        myService1: myService1,
        myService2: myService2
        // [...]
    };
}

// Usage - File `main.js` or in another module !
// [...]
var myModule = myConstructor(dependency1, dependency2);
myModule.myService1(option1, option2);
```

2.F. Module Vanilla IV - Avec dépendances dynamiques

Un exemple avec un service de sortie d'écran.

```
// Definition - File `output.js`
function outputConstructor(media) {
    "use strict";

    // [...] The module code
    var buffer = [];

    function push(value) {
        buffer.push(value);
        return this;
    }

    function output() {
        media(buffer.join(' '));
        buffer = [];
    }

    // Expose the service in the ask set
    return {
        push: push,
        output: output
    };
}

// File `main.js` or in another module !
// [...]
var logger = {};
logger.log = outputConstructor(console.log);
logger.error = outputConstructor(alert);
logger.log.push('Brown').push('Bag').push('Lunch').output();
logger.error.push('Isolated').push('JavaScript').push('FTW').output();
```

2.G. Module Vanilla V - Avec dépendances statiques

Un modèle pour gérer les dépendances statiques.

```
// Definition - File `myModule.js`
var myConstructor = (function myDeclaration(dependency1, dependency2) {
    "use strict";

    return function myConstructorF(argument1, argument2) {

        // [...] The module code
        function myService1() {}

        function myService2() {}

        // Return a set of function
        return {
            myService1: myService1,
            myService2: myService2
            // [...]
        };
    }
})(dependency1, dependency2);

// Usage - File `main.js` or in another module !
// [...]
var myModule = myConstructor(argument1, argument2);
myModule.myService1(option1, option2);
```

2.G. Module Vanilla V - Avec dépendances statiques

Un exemple avec un service de sortie d'écran.

```
1  // Definition - File `output.js`
2  var outputConstructor = (function outputDeclaration(media) {
3      "use strict";
4
5      return function outputConstructorF(prefix) {
6          // [...] The module code
7          var buffer = [];
8
9          function push(value) {
10             buffer.push(value);
11             return this;
12         }
13
14         function output() {
15             buffer.unshift(prefix);
16             media(buffer.join(' '));
17             rule();
18             buffer = [];
19         }
20
21         function rule() {
22             media('----------------');
23         }
24
25         // Expose the service in the ask set
26         return {
27             push: push,
28             output: output
29         };
30     }
31 })(console.log);
32
33 // File `main.js` or in another module !
```

2.H. Module AMD I - Avec RequireJS

Un modèle pour gérer des arbres de dépendances.

```
// Definition - File `myModule.js`
define(['dependency1', 'dependency2'], function myConstructor(dependency1, dependency2) {
    "use strict";

    // [...] The module code
    function myService1() {
    }

    function myService2() {
    }

    // Return a set of function
    return {
        myService1: myService1,
        myService2: myService2
        // [...]
    };
});

// Usage - File `main.js`
// [...]
var myModule = require("myModule");
```

2.H. Module AMD I - Avec RequireJS

Un exemple avec un service de sortie d'écran.

```
// Definition - File `output.js`
define("output", [], function outputConstructor() {
    "use strict";

    // [...] The module code
    var buffer = [];

    function push(value) {
        buffer.push(value);
        return this;
    }

    function output() {
        console.log(buffer.join(' '));
        buffer = [];
    }

    // Expose the service in the ask set
    return {
        push: push,
        output: output
    };
});

// Usage - In another module
require(['output'], function (logger) {
    logger.push('Brown').push('Bag').push('Lunch').output();
});

// Usage - File `main.js`
require(['require'], function (require) {
    var logger = require('output');
    logger.push('Isolated').push('JavaScript').push('FTW').output();
});
```

2.I. Module AMD II - Avec RequireJS

Un modèle AMD avec multi-instances.

```
// Definition - File `myModule.js`
define(['dependency1', 'dependency2'], function myDeclaration(dependency1, dependency2) {
    "use strict";

    return function myConstructor(argument1, argument2) {

        // [...] The module code
        function myService1() {}

        function myService2() {}

        // Return a set of function
        return {
            myService1: myService1,
            myService2: myService2
            // [...]
        };
    };
});

// Usage - File `main.js`
// [...]
var myConstructor = require("myModule");
var myModule = myConstructor(argument1, argument2);
```

2.I. Module AMD II - Avec RequireJS

Un exemple avec un service de sortie d'écran.

```
// Definition - File `output.js`
define("multi-output", [], function outputDeclaration() {
    "use strict";

    return function outputConstructor(media) {

        // [...] The module code
        var buffer = [];

        function push(value) {
            buffer.push(value);
            return this;
        }

        function output() {
            media(buffer.join(' '));
            buffer = [];
        }

        // Expose the service in the ask set
        return {
            push: push,
            output: output
        };
    }
});

// Usage - In another module
require(['multi-output'], function (loggerConstructor) {
    var logger = loggerConstructor(console.log);
    logger.push('Brown').push('Bag').push('Lunch').output();
});
```

3. jQuery plugins

3.A. Pourquoi utiliser des plugins jQuery ?

Le but principal est de limiter / d'améliorer la maintenance :

1. **Lier facilement du code à un élément DOM** ;
2. Réduire la quantité de coder en utilisant les APIs jQuery ;
3. Augmenter la lisibilité ;

3.B. Plugin jQuery I - Avec multi-éléments

Un modèle pour commencer.

```
// Definition - File `jquery.myPlugin.js`
(function ($) {
    $.fn.myPlugin = function () {
        return this.each(function () {
            // [...] The plugin code where `this` is every element
        });
    };
})(jQuery);

// Usage - File `main.js`
// [...]
jQuery('#query').myPlugin();
```

3.B. Plugin jQuery I - Avec multi-éléments

Un exemple avec les liens de partage. **Tweet** !

```
// Definition - File `jquery.sharePopup.js`
(function ($) {
    $.fn.sharePopup = function () {
        this.each(function () {
            var $button = $(this);
            $button.on('click', function (event) {
                event.preventDefault();
                window.open($button.attr('href'), "", "width=640, height=280");
            });
        });
        return this;
    };
})(jQuery);

// Usage - File `main.js`
// [...]
jQuery('.twitter1').sharePopup();
```

3.C. Plugin jQuery II - Avec options

Un modèle qui permet d'avoir des options.

```
// Definition - File `jquery.myPlugin.js`
(function ($) {
    var defaultOptions = {
        // [...] The plugin default options
    };
    $.fn.myPlugin = function (options) {
        options = $.extend({}, defaultOptions, options);
        return this.each(function () {
            // [...] The plugin code with `this` is every element
        });
    };
})(jQuery);

// Usage - File `main.js`
// [...]
jQuery('.query').myPlugin({option1: value1, option2: value2});
```

3.C. Plugin jQuery II - Avec options

Un exemple avec les liens de partage. **Tweet** !

```
// Definition - File `jquery.sharePopup.js`
(function ($) {
    var defaultOptions = {
        width: 640,
        height: 280
    };
    $.fn.sharePopup = function (options) {
        options = $.extend({}, defaultOptions, options);
        this.each(function () {
            var $button = $(this);
            $button.on('click', function (event) {
                event.preventDefault();
                window.open($button.attr('href'), "", "width=" + options.width + ", height=" + options.height);
            });
        });
        return this;
    };
})(jQuery);

// Usage - File `main.js`
// [...]
jQuery('.twitter2').sharePopup({height: 400});
```

3.D. Plugin jQuery III - Avec sous-actions

Un modèle qui permet de gérer des sous-actions.

```
// Definition - File `jquery.myPlugin.js`
(function ($) {

    var defaultOptions = {
        // [...] The plugin default options
    };
    var pluginConstructor = function myPlugin($container, options) {
        "use strict";

        // [...] The module code with `$container` is every element

        function myService1() {}

        function myService2() {}

        // Return a set of function
        return {
            myAction1: myService1,
            myAction2: myService2
            // [...]
        };
    };

    // jQuery plugin encapsulation
    var pluginName = pluginConstructor.name;
    $.fn[pluginName] = function (action) {
        if (typeof action !== 'string') {
            var options = $.extend(true, {}, defaultOptions, action);
            this.each(function () {
                this[pluginName] = pluginConstructor($(this), options);
            });
        } else {
            var args = Array.prototype.slice.call(arguments, 1);
```

3.D. Plugin jQuery III - Avec sous-actions

Un exemple avec les liens de partage. **Tweet** !

```
// Definition - File `jquery.sharePopup.js`
(function ($) {
    var defaultOptions = {
        width: 640,
        height: 280
    };
    var pluginConstructor = function sharePopup($button, options) {
        "use strict";

        $button.on('click', function (event) {
            event.preventDefault();
            window.open($button.attr('href'), "", "width=" + options.width + ", height=" + options.height);
        });

        function open() {
            $button.click();
        }

        return {
            open: open
        }
    };

    // jQuery plugin encapsulation
    var pluginName = pluginConstructor.name;
    $.fn[pluginName] = function (action) {
        if (typeof action !== 'string') {
            var options = $.extend(true, {}, defaultOptions, action);
            this.each(function () {
                this[pluginName] = pluginConstructor($(this), options);
            });
        } else {
            var args = Array.prototype.slice.call(arguments, 1);
```

3.E. Plugin jQuery IV - Sans la structure jQuery

Un modèle qui permet d'avoir une gestion globale, et non *n* instances indépendantes.

```
// Definition - File `myModule.js`
var myConstructor = (function myDeclaration($) {
    "use strict";

    var defaultOptions = {
        // [...] The plugin default options
    };

    return function myConstructorF(selector, options) {

        options = $.extend(true, {}, defaultOptions, options);

        // [...] The module code with `selector` to query elements

        function myService1() {}

        function myService2() {}

        // Return a set of function
        return {
            myService1: myService1,
            myService2: myService2
            // [...]
        };
    }
})(jQuery);

// Usage - File `main.js`
// [...]
var myModule = myConstructor('.twitter4', {option1: value1});
// [...]
myModule.myService1(argument1, argument2);
```

3.E. Plugin jQuery IV - Sans la structure jQuery

Un exemple avec les liens de partage. **Tweet** !

```
// Definition - File `sharePopup.js`
var sharePopupConstructor = (function sharePopupDeclaration($){
    "use strict";

    var defaultOptions = {
        width: 640,
        height: 280
    };

    return function sharePopupConstructorF(selector, options) {

        options = $.extend(true, {}, defaultOptions, options);
        $(document).on('click', selector, function (event) {
            event.preventDefault();
            openLink($(this));
        });

        function open() {
            openLink($(selector));
        }

        function openLink($link) {
            window.open($link.attr('href'), "", "width=" + options.width + ", height=" + options.height);
        }

        return {
            open: open
        }
    }
})(jQuery);

// Usage - File `main.js`
// [...]
```

☺ Merci

Des questions, des retours ?

- **hello@tzi.fr**
- **@iamtzi**
- **tzi.fr**

Revoir cette présentation en ligne **git.io/ModulesJS**.