# Une présentation en orienté objet

Je suis parti d'une idée simple, et si les slides n'étaient que des objets dont il suffit de modifier les attributs pour la manipuler. C'est que cette version de l'Editor Jmpress propose.

## Un peu (très peu) de vocabulaire

### Composants, slides et élements

La présentation est composée de slide contenant des élements qui peuvent être du texte, des images ou n'importe quoi qui peut s'afficher en HTML/CSS sans plugin. L'ensemble des slides et des élements sont des composants.

### Matricules

Chaque composant est repéré par un matricule unique. Par défaut la matricule s'appuie sur le type du composant. Un compteur global se charge de l'unicité. Néanmoins, tant que le matricule est unique, il est possible de renseigner ses propres matricules.

## Les différentes classes

Chaque entités permettant de créer la présentation est une instance d'une classe, dotées d'attributs et de méthodes.

### La classe Slide

### Creation

La classe la plus importante dans la mesure où il s'agit d'un éditeur pour une présentation. La manipulation (création/édition/suppression) des slides se fait donc en objet, comme un langage complétement objet tel que le Java.

Ainsi pour créer une slide vous ferez :

var slide = new Slide();

et une slide vide sera crée et apparaitra dans la présentation avec des coordonnées par défaut. Une slide est défini par un matricule unique, une position dans l'espace, une orientation également dans l'espace ainsi qu'une taille. Ses paramètres constitues les attributs de la classe Slide dont chaque slide est une instance. Les attributs sont eux-mêmes des objets javascript, les voici :

* matricule: //par défaut, slide+cptcomposant
* pos : {

x:

y :

z :

}

* rotate : {

x:

y :

z :

}

* propeties : {

scale :

hierarchy :

}

Vous pouvez créer une slide directemement avec des attributs, pour cela il faut passer un seul argument à new Slide();, cet argument étant un objet contenant les atributs objets que vous voulez renseigner.

Par exemple, pour créer une slide avec des coordonnées :

var slide = new Slide(

pos : {

x : 200,

y : 400

}

);

Idem pour créer une slide avec des rotations :

var slide = new Slide(

rotate : {

y : 45

}

);

Comme vous pouvez le constater, il est possible de ne renseigner qu'une seule coordonnée, le constructeur de la classe Slide se chargera de renseigner les autres paramètres avec les valeurs par défaut.

Note : Les rotates sont en degrés, ainsi renseigner rotate.x : 360 n'aura pas d'effet visible.

Notes : les valeurs par défaut sont 0 que ce soit pour la position ou pour la rotation

### Modification

C'est bien beau de créer des slides où nous voulons, orientées comme nous voulons, mais le but d'un éditeur est bien de proposer de déplacer et d'orienter des slides à convenances. C'est très facile !

Prenez votre slide créer via :

var slide = new Slide();

Obtenir sa position ?

var posX = slide.pos.x;

Modifier sa position ?

slide.pos.x = 1000;

Modifier sa rotation ?

slide.rotate += 20; //en degrés, sens anti-trigo

Modifier sa taille ?

slide.properties.scale = 10; //scale = 1 est une slide de 700\*900, le scale est proportionel

### Suppression

Supprimer une slide est aussi simple que :

slide.destroy();

### Voir dans console

Parce que parfois il peut être utile de voir les attributs de la slide,

slide.show();

permet d'afficher en console l'ensemble des attributs (pos, rotate, properties) ainsi que le nombre d'element contenus dans la slide

slide.show('element');

permet d'afficher en console l'ensemble des attributs des éléments contenus dans la slide.

En parlant d'élements de slide, voici ce qu'ils sont.

### La classe Element et ses enfants

Les élements composants les slides fonctionnent sur le même principe que les slides à ceci près que le constructeur exige en premier paramètre le matricule d'une slide dans laquel insérer l'element. En effet, un element n'a pas d'éxistence en dehors d'une slide, lors de sa création il faut renseigner une slide de destination. Mais comme je suis gentil, si jamais vous ne savez pas où insérer un element ou si le matricule que vous renseigné ne correspond pas à une slide éxistante, le constructeur proposera à l'utilisateur de sélectionner une slide via un click (le fond de la présentation est alors assombri jusqu'à ce qu'une slide soit selectionée par un click).

La création et l'édition des élements suivent la même logique que les slides. A la création vous pouvez renseigner en second parametre un objet Javascript contenant les attributs objets que vous voulez renseingner. Pour la modification le principe est le meme qu'avec les instances de Slide, vous modifier l'instance d'un element et c'est répercuté sur la présentation.

Attention, Element est une classe interface, instancier un Element ne posera pas d'erreurs mais ne servira à rien !

Pour ne pas me repeter, je vais juste détailler les atributs et les méthodes de chaque enfant de Element qui peuvent être définis lors de la création ou modifier via l'instance comme pour slide.

#### Texte

* matricule: //par défaut, elementtexte+cptcomposant
* pos : {

x:

y :

}

* properties : {

content : //contenu sous forme de texte

hierarchy : // bodyText, H1Text, H2Text, H3Text

}

* show()
* destroy()

note : pour éditer du texte via l'interface, double click sur le texte en question, le reste est intuitif.

#### Image

* matricule: //par défaut, image+cptcomposant
* pos : {

x:

y :

}

* size : {

height:

width:

}

* source : //source HTML ou adresse fichier de l'image
* show()
* destroy

Voilà comment créer un élement dans une slide (via son matricule). Bien que ce soit tentant, il n'est pas actuellement (4 aout 2013) possible de créer une slide avec directement ses elements. Pour ce faire il faut créer une slide, récupérer son matricule et ensuite créer les élements dans le matricule.

var slide = new Slide();

var matricule = slide.matricule;

new Text(matricule);

ceci créera une slide au centre du monde des slides avec un texte en corps de texte environs au centre de slide avec comme contenu : "Type texte here"

### La classe Transform3D

Une fois la création et la modification maitrisée, il faut pouvoir se déplacer dans la présentation, une classe est prévue pour cela. Elle se manipule de la même manière que les instances des autres classes (Slide, Text, Image) à ceci près qu'il n'éxiste qu'une seule instance pour gérer la navigation, cette instance est initialisée lors du chargement de l'Editor JmpressV2 est se nomme :

transform3D

ses attributs sont les suivants :

* pos : {

x:

y :

z :

}

* rotate : {

x:

y :

z :

}

Ils permettent de se déplacer en 3 dimensions dans la présentation (appelé monde des slides).

## Techno utilisées

### SimpleJsInheritance.js

Javascript est un langage objet, on accède aux méthodes d'une instance array en mode objet (par exemple tableau.push(valeur) ). Malheureusement, le Javascript ne permet pas nativement de créer et de manipuler soit même ses classes et ses méthodes. Plusieurs solutions existent pour combler ce manque. Celle que j'ai choisi se nomme Simple Js Inheritance et permet, via un script commentés et aérés de 25 lignes, de créer des Classes contenant des méthodes et autorisant l'héritage.

Sa documentation est disponible ici :

<http://ejohn.org/blog/simple-javascript-inheritance/>

Elle est très courte mais permet de donner accès au concept de Classe, de méthodes et d'héritages dont nous avons l'habitude avec des langages tel que Javascript.

Les concepts évolués tels que les interfaces ne sont pris en compte que la rigueur du codeur. Par exemple, ma classe Element est une interface mais rien n'empèche de l'instancier, si ce n'est pas que ca n'aura aucuns effets.

Ce mini script de 25 lignes se concentre sur l'essentiel de l'héritage et laisse le soin aux développeurs d'être rigoureux quant à son usage.

D'un point de vue technique, Simple JS Inheritance se base sur un objet javascript dont les attributs objets sont les attributs de classe et les attributs fonctions sont les méthodes de la classe (en tirant profit des posibilités des "prototype" de chaque objet permit par JS).

L'ensemble de mes classes tirent partie de la puissance de cet outil merveilleux.

Pour ceux ne voulant pas lire la documentation (très courte) de Simple Js Inheritance, les construceurs sont dans

init : function(….) {

}

les attributs de la classe sont des clef-valeur en objet de l'objet simili class,

this.pos = {

…

}

les méthodes sont des clef-valeur de l'objet contenant en valeur une fonction (comme le construteur)

show: function() {

console.log(…);

}

Concernant l'héritage, coici un appel au constructeur de la mère (ici, class Element) en lui passant des paramètres.

this.\_super(slide, params, matricule)

Toute cette mécanique d'héritage en 25 lignes, merci John Resig !

### Watch.js

Bon, nos composants sont désormais des instances de classes, dotés d'attributs et de méthodes. Pour la modification des attributs des composants j'aurais pu faire des méthodes set pour chaque attributs qui se chargent de faire la modification dans le DOM.

En effet, les instances de Slide ou de Text ne sont que des images en mémoire des slides et des textes éxistants dans le DOM que l'user voit à l'écran.

Via des set, la syntaxe aurait été :

slide.setPos('x',100);

ou encore

slide.setPos('x',slide.get('x')+100);

penible… Alors que ce que je propose c'est de faire :

slide.pos.x += 100;

Directement modifier les attributs de l'objet (donc l'instance) et que cela mette à jour le DOM, cela est rendu possible via une très intéressant technologie, le watch.

Le watch est une nouvelle méthode des objets Javascript qui permet de détecter les changements dans un objet Javascript. Un peu comme un listerner 'onchange' sur un élement du DOM. Malheuresement cette fonctionnalité n'est actuellement (aout 2013) supportée que par Firefox. Afin de rendre Editor JmpressV2 le plus cross-plateform possible, il a recourt à Watch.js, un script qui rend accessible le watch à n'importe quel navigateur interpretant du Javascript.

Les constructeurs des classes Slide et Element utilisent des watch de Watch.js pour detecter les modifications des attributs des instances correspondantes et effectuer le traitement adéquat. Par exemple, la modification de la coordonée d'une slide entraine la modification de la coordonée dans le DOM ainsi que la desinit puis réinit de la slide via Jmpress. A l'inverse, la modification de la coordonnée d'un Element (tel que le Text) n'entraine que la modification des atrinuts CSS (top et left) directement dans le DOM. Le navigateur se chargeant de répercuter les changements lors de l'affichage.

Un watch via Watch.js respecte la syntaxe du watch proposé par le W3C à ceci près que la watch du W3C est une méthode des objets alors que le watch de Watch.js est une fonction prenant en paramtrère l'objet à scruter puis la fonction (callback) à réaliser lors de la modification de l'objet scruté.

Voici un exemple de watch sur l'attribut objet 'pos' issu d'un Element (donc, modification dans le DOM suffit):

Watch dans le constructeur de la classe (interface, doit être instanciée par une fille) Element

watch(this.pos, function(attr, action, newVal, oldVal) {

//mise à jour du DOM

var $element = $('#' + matricule); //recupération du matricule de l'élement modifié

var attribut;

switch (attr) { //detection de l'attribut mis à jour

case 'x':

attribut = 'left';

break;

case 'y' :

attribut = 'top';

break;

default :

return;

}

$element.css(attribut, newVal); //modification du CSS de l'element concerné

});

Autre exemple, le watch de la position d'une slide. Celui là est plus complexe car en plus de mettre à jour le DOM il doit demander à Jmpress de reprendre en considération la nouvelle slide (desinit, init):

watch(this.pos, function(attr, action, newVal, oldVal) {

//mise à jour du DOM

var $slide = $('#' + matricule); //recupération de la matricule

$('#slideArea').jmpress('deinit', $slide); //jmpress desinit la slide

var attribut = 'data-' + attr;

$slide.attr(attribut, newVal); //mise à jour du DOM

$('#slideArea').jmpress('init', $slide); //Jmpress réinit la slide avec ses nouvelles coordonées

});

Pas bien plus compliqué n'est ce pas ?

Pour conclure, avec ceux deux technos nous rencontrons les limites du langage Javascript; vieux et pas du tout destiné à ce que nous voulons en faire. Mais en même temps nous rencontrons la volonté de la communauté de sublimer de langage en utilisant des artifices et en exploitant ses fonctionnalités au possible afin de proposer un langage puissant et au moins à la hauteur de n'importe quel langage plus récent.

Javascript, via sa communauté infinie et passionnée par le libre, à de très beaux jours devant lui. J'adresse ici un immense merci à ceux qui y travaillent chaque jour.

# Les interfaces utilisateurs

Grâce aux technologies misent place plus haut, il est possible de manipuler les instances des clases Slides ou des filles de Javascript ou encore de Transform3D via la console. Les slides étant contenues dans la variable globale *container*.

L'idée de cette manipulation facile d'accès est de rendre très simple le développement d'interface utilisateur. Afin de rendre les intéractions encore plus simple j'ai mis au point un objet, *objectEvent* qui permet de soumettre encore plus simplement des actions aux instances de classes. Avec cet objet qu'on manipule comme des chaussettes, il n'est même pas nécessaire de comprendre comment fonctionne les instances des classes (pourtant c'est ultra facile !).

## ObjectEvent

Il existe une classe ObjectEvent permettant d'effectuer simplement des actions sur les instances des composants ainsi que de Transform3D. ObjectEvent est également une classe qui s'instancie, ses attributs minimum sont :

* matricule : str
* action : str
* event : obj

Matricule contient le matricule du composant à modifier, si c'est un string vide cela signifie qu'il s'agit de la création de composant.

Action représente l'action à effectuer sur le composant tel que le déplacement (move) ou la rotation (rotate), ou bien la description du type de composant à créer (createBodyText).

Pour comprendre l'utilité de cette classe, prenons le cas de la création d'une slide. Pour le moment (aout 2013) une slide peut être créé via le bouton de l'interface graphique ('Slide') ou bien en utilisant la touche 'k' lorsque la souris ne survole aucuns composants. Les deux méthodes instancient le même objectEvent (puisqu'elles font la même chose). L'exemple des slides est trivial, c'est bien plus simple de faire new Slide(); que d'instancier ObjectEvent et d'appeler une fonction avec l'instance en paramètre. Mais prenons les éléments, ils ne peuvent exister en dehors d'une slide il faut donc une mécanique avant la création pour savoir où insérer l'element. C'est le cas du texte, il suffit d'instancier un ObjectEvent avec en action 'createBodyText' et en attribut 'target' le matricule de la slide. Ensuite une routine se chargera de vérifier si le matricule passé est bien celui d'une slide, si ce n'est pas le cas, elle lancera d'autres routines qui inviteront l'utilisateur à selectionner une slide.

Pour le coup, instancier ObjectEvent est bien plus court !

Je détaillerai le fonctionnement des routines qui 'lisent' objectEvent dans la prochaine partie, ci après je détaille l'ensemble des différentes manières d'instancier ObjectEvent déjà prise en charge.

Manipulation de l'interface en elle-même, par oppostion à la manipulation de la présentation. Ici on parle de création de composants ou bien de déplacement dans la présentation

matricule : '' OU matricule: 'document'

action: 'createImage' OU 'createBodyText' OU 'createH3Text' OU 'createH2Text' OU 'createH1Text'

## L'interface de création/gestion

La plus simple à comprendre est la gestion par le clavier.

////detechtion du hover + keyboard

## La gestion à la souris

////outils joystick

### Boutons

////conection interface au create

### Timeliness

Methode reOrder

### Sauvegarde

### Mode présentation

# Organisation du code

/// differents fichiers

/// localisation de chaque bout de code

## Dictionnaire des classes et de leurs méthodes

## Dictionnaire des objets

### Dictionnaire des fonctions d'aide

# Have fun, simply !

## La puissance de la simplicité

## La suite