

Haxetelier 8

Présentation

Jean-Michel Delettre.

Développeur d'applications web.

- Flash
- puis Html5.

Les exemples sont en

Haxe  compilé  JS

Haxe étant multi-cibles
les exemples sont transposables

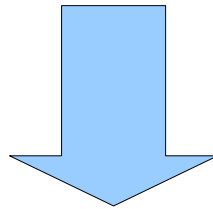
Préambule pour $\overline{\text{hx}}$

Haxe a un formalisme typique/classique POO :

- Une classe est un bloc continu, commençant par le mot-clé « class » (!= fonction)
- L'héritage se fait avec le mot-clé « extends » (!= prototype)
- Les variables et propriétés sont fortement typées ; ainsi que le retour des méthodes.

[Typage de variables

- Toutes les variables ont un type (Int, String, MyClass, Date, etc)
déclaré explicitement ; ou implicitement lors de la 1ère assignation
- A la compilation, Haxe refusera qu'elles reçoivent un contenu d'un autre type ! Le paramètre de retour des fonctions est aussi vérifié.



On va considérer que c'est un gros avantage
qui corrige plein de coquilles, d'erreurs bêtes
... et parfois moins bêtes.

« using »

Finalité :

Ajouter des méthodes à une classe existante
quand justement on ne peut pas l'étendre !

```
someNativeInstance.myMethod() ;
```

Exemples :

"Element" de l'api JS et "MovieClip" de l'api Flash
déjà instancié, respectivement dans le html ou le swf.

Les "String" en dur

...etc.

« using » suite

Exemple avec « String »

où les chaines de caractères contiennent de la syntaxe CSS

".someCssClass p"

"#myCtnr #myDivId"

"div.menu , ul.menu"

.

« using » suite

Méthodes ajoutées :

get() :

```
"#myDivId".get().appendChild(elem) ;
```

```
"#myCtnrId .h1".get().textContent = someTitle ;
```


« using » suite

Méthodes ajoutées (suite):

`all()`

`".someCssClass".all();`

« using » suite

Méthodes ajoutées (suite) :

`on()` => équivalent de `addEventListener`

`"div.menu".on("click",someListener,false,someObject) ;`

et

`off()`

« using » suite

Méthodes ajoutées (suite) :

slider() :

au lieu de

```
var s = new Slider();  
s.into = "#sliderCtnrld" ;
```

on a:

```
"#sliderCtnrld".slider() ;
```

haxetelier8 complet :

<https://github.com/flashline/haxetelier8> > [download zip]

Haxe :

<http://haxe.org/download/>


« using » suite

Ouvrir :

`samples/using/src/Main.hx`

Exemple d'abstraction avec « using »
quand on cible plusieurs 'api' (js,flash,etc) :

```
#if js
    import js.html.Element;
    using apix.common.display.ElementExtender;
    typedef Elem = Element ;
#else if flash
    import flash.display.Sprite;
    using apix.common.display.SpriteExtender;
    typedef Elem = Sprite ;
    // to continue !
#else
    // TODO
#endif
```



// utiliser **Elem** dans le code
// avec les méthodes des **xxxExtender** (ou méthodes communes si existent).

Sources :

sample/classes/Apix/apix/ui/slider/[Slider.hx](#)
sample/classes/Apix/apix/common/display/[ElementExtender.hx](#)

« extern »

Finalité :

Utiliser une API native du langage cible,

ex : **jQuery**

... sans avoir à l'implémenter en haxe,

tout en bénéficiant de ses avantages dont le

typage fort !

Plein d' « extern » sur : <http://lib.haxe.org/all>

« extern »

Principe avec une API 3D en JS :

Babylon*

* de David Catuhe @deltakosh <http://www.babylonjs.com/>

download de l'extern : <https://github.com/flashline/Babylon-X>

"extern" suite

Voir :

<samples/extern/bin/meshTransform.html>

"extern" suite

Démos:

<http://www.pixaline.net/intra/3djs/>

FIN

...prochain sujet : Apache [Cordova](#) (~ [PhoneGap](#))

Haxe +