Une introcution pragmatique à Seaside

— Le framework en Smalltalk pour développer des applications Web sophistiquées —

Luc Fabresse

luc.fabresse@mines-douai.fr

Plan

Plan

Qu'est-ce que Seaside ?

Un framework écrit en Smalltalk pour le développement d'applications Web

Pourquoi est-ce intéressant pour vous ?

- Étudier une technologie alternative (autre que PHP, Java, Flash/Flex)
- Découvrir un framework innovant

Pré-requis

Le développement Web

- HTTP
- HTML, CSS
- Formulaires, POST, GET, ...
- Persistance

Le langage Smalltalk

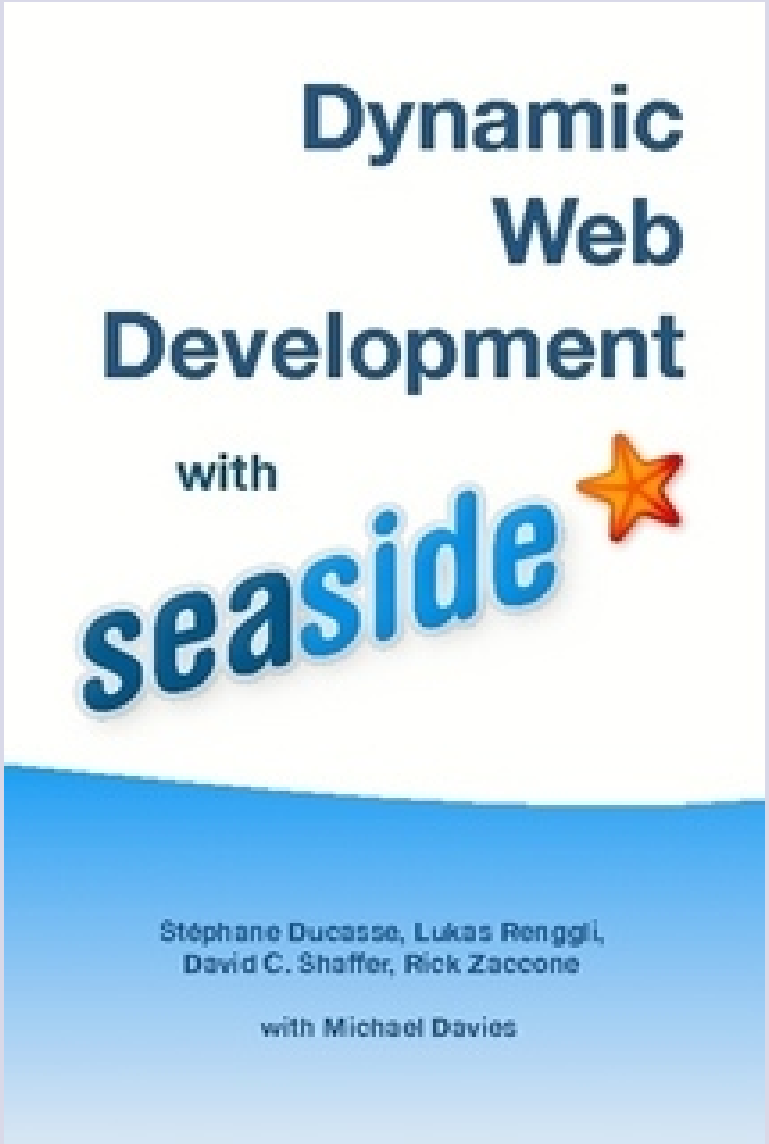
- Langage orienté objets
- Dynamiquement typé
- Réflexif

Bibliographie (1)

http://www.seaside.st

Bibliographie (2)

http://book.seaside.st



Démo `WACounter`

Plan

Notion de *composant*

Qu'est-ce qu'un *composant* ?

- un élément visuel d'une page HTML (généralement une `div`)
- une instance d'une sous-classe de `WAComponent`

Notion de *composant principal*

Toute application Web en Seaside possède un composant principal

Générer du XHTML

Affichage d'un composant en XHTML :

Un composant «sait s'auto-afficher» en HTML via sa méthode `renderContentOn:` (héritée de `WAComponent`). Cette méthode est automatiquement appelée par Seaside.

La méthode `renderContentOn:`

prend un paramètre (nommé `html`) instance de `WARenderCanvas`

La classe `WARenderCanvas`

fournit une interface permettant de facilement générer du XHTML valide en utilisant des « pinceaux »

Les « pinceaux » XHTML

Le pinceau paragraph

```
html paragraph: 'a simple text'.
html paragraph
   with: 'a simple text'
```

Le pinceau div

```
html div: 'a simple text'.
html div
   with: 'a simple text'
```

Utilisation combinée

```
html div
   with: [html paragraph: [ 'a simple text' ].
      html image url: 'http://www.seaside.st/styles/logo-plain.png']
```

Le pinceau universel render:

```
MyComponent>>renderContentOn: html
   html render: 'une chaine'.
   html render: anotherComponent
```


En résumé

- Un composant `MyApp` est une sous-classe de `WAComponent`
- Pour qu'un composant puisse être une application (point d'entrée), ajouter la méthode :

```
MyApp class>>canBeRoot
    ^true
```

- Pour que ce composant affiche du HTML, définissez la méthode :

```
MyApp>>renderContentOn: html
    html text: 'My App is so cool'
```

- Utiliser les pinceaux
  `http://book.seaside.st/book/fundamentals/rendering-components/learning-canvas-and-brush`

A propos de `renderContentOn:`

- Ne fait qu'afficher
- Pas d'instanciation, pas de modification, ...

Plan

Le message class: des pinceaux

```
html div
   class: 'center';
   with: 'Seaside is cool'.
html paragraph
   class: 'highlight';
   with: 'Highlighted text'
```

Le message class:if: des pinceaux

```
html unorderedList with: [
   1 to: 10 do: [ :index |
      html listItem
         class: 'even' if: index even;
         with: index ] ]
```


Gestion du code CSS

Solution 1 : la méthode `style` des composants

```
MyComponent>>style
   ^ '.even { background-color: light-gray;}'
```

Solution 2 : un fichier CSS unique

1. dans un fichier externe (`<link ...`)

```
MyComponent>>updateRoot: anHtmlRoot
   super updateRoot: anHtmlRoot.
   anHtmlRoot stylesheet url: 'http://car.ensm-douai.fr/site21/mystyle.
```

2. dans une `FileLibrary` (sous classe de `WAFileLibrary`)

```
MyFileLibrary addAllFilesIn: '/path/to/directory'
MyFileLibrary addFileAt: '/path/to/mystyle.css'

MyComponent>>updateRoot: anHtmlRoot
   super updateRoot: anHtmlRoot.
   anHtmlRoot stylesheet url: MyFileLibrary / #mystyleCss
```


Autre bibliothèques CSS

http ://pharo.pharocloud.com/

Plan

Le pinceau anchor

Affichage d'un lien

```
MyComponent>>renderContentOn: html
   html anchor
      url: 'http://www.seaside.st';
      with: 'Seaside Website'
```

Notion de callback

Un block (fonction anonyme) qui sera executé *automatiquement* lorsque l'ancre sera cliquée. Pas d'affichage (le renderer html est invalide) dans un callback.

Exemple de callback

```
MyComponent>>renderContentOn: html
   html anchor
      callback: [ self clicked ] ;
      with: 'You already clicked me ', number, ' times'
MyComponent>>clicked
   number := number + 1
```


Démo `WAMultiCounter`

Plan

Edition des données d'un utilisateur

```
UserEditorComponent>>renderContentOn: html
    html form: [
        html text: 'Login: '.
        html textInput
            callback: [ :value | self user login: value ];
            value: self login name.
        html break.
        html text: 'Email address: '.
        html textInput
            callback: [ :value | self user emailAddress: value ];
            value: self user emailAddress.
        html break.
        html submitButton
            callback: [ self save ];
            value: 'Save']
UserEditorComponent>>save
self inform: self user login , '--' , self user emailAddress
```


Gestion simplifiée des formulaires

`http://book.seaside.st/book/fundamentals/forms/convenience`

Une application exemple : Gestion d'une todo-list
`http://book.seaside.st/book/in-action/todo`

Plan

Le mécanisme Call/Answer

Documentation : `http://book.seaside.st/book/components/calling`

Plan

Scriptaculous

Documentation+démo : `http://scriptaculous.seasidehosting.st/`

JQuery

Documentation+démo : `http://demo.seaside.st/javascript/jquery`

JQueryUI

Documentation+démo : `http://demo.seaside.st/javascript/jquery-ui`

Plan

En résumé

Concepts et mécanismes innovants en Seaside :

- un modèle de *composants* et *callbacks*
- du code HTML toujours valide
- les objets métiers sont toujours présents (paramètres GET/POST gérés automatiquement)
- possibilité de débugguer facilement les applications Web
- encapsulation de tehchnologies/frameworks Web 2.0 : AJAX, JQuery, ...

Alors ?