VEGAS - Design Pattern d'inversion de contrôle : Chargement de ressources externes

par Marc Alcaraz (Mon site)

Date de publication : 11/09/2009

Dernière mise à jour :

J'attaque (enfin) le dernier article sur les ressources externes chargées pendant l'initialisation d'une fabrique **IoC**. CeT article va me permettre de terminer la série d'articles sur l'implémentation **IoC** de **AndromedAS**.

Pour ceux qui découvrent, ici, le design pattern d'Inversion de contrôle avec **VEGAS** et qui n'ont pas eu encore le temps de consulter les articles précédents, voici une liste des articles à lire avant d'entamer celui-ci :

- **VEGAS - Design Pattern d'inversion de contrôle : Généralités**
- **VEGAS - Design Pattern d'inversion de contrôle : Définitions d'objets**
- **VEGAS - Design Pattern d'inversion de contrôle : Configuration d'une fabrique IoC**

I - Généralité

Un objet "**import**" peut être défini dans tous les fichiers externes de configuration et d'initialisation d'une fabrique **IoC**.

Cet objet est facultatif et sera interprété par les chargeurs basés sur la classe **andromeda.ioc.net.ObjectFactoryLoader**, il contient des **objets génériques** simples qui permettent de définir une collection de différentes ressources externes qui seront chargées pendant l'initialisation de la fabrique et utilisées ensuite pendant la création des définitions d'objet mais aussi à n'importe quel autre moment dans le cycle de vie de l'application.

I-A - Utilisation de plusieurs fichiers de configuration pour initialiser le conteneur léger.

Le fichier de configuration principal chargé par la classe **ECMAObjectLoader** peut être découpé en plusieurs sous fichiers chargés récursivement les uns dans les autres.

Pour charger des sous fichiers de configuration dans un fichier externe, il faut insérer à la base de l'objet principal contenu dans le **DOM** de configuration un attribut "**imports**" de type **Array** qui contiendra une suite d'objets génériques qui définissent les fichiers que l'on souhaite charger.

Exemple

```
imports =
[
    { resource : "logging.eden"     } ,
    { resource : "view.eden"        } ,
    { resource : "controller.eden" }
] ;
```

Chaque fichier chargé ci-dessus contiendra lui aussi des **définitions d'objet** ou d'autres définitions de ressources qui permettront d'initialiser le conteneur léger et l'application. Cette utilisation basique pour charger des ressources au format **eden** est en quelque sorte la partie immergée de l'iceberg car il est possible d'aller beaucoup plus loin avec l'attribut "**import**".

I-B - Importation de différents types de ressources.

Pour initialiser un conteneur **IoC**, il n'est pas suffisant de pouvoir définir seulement ses **définitions d'objet**. Une application nécessite généralement le chargement de plusieurs éléments graphiques, feuilles de styles, objets de configuration simples ou des fichiers permettant de localiser l'application, sons, polices de caractères, images, animations etc.

Tous ces éléments peuvent être inclus directement dans le **swf** principal de l'application mais à force cette centralisation dans un seul fichier peut devenir très contraignante.

En effet si tous ces éléments sont confinés dans le **swf** principal nous sommes rapidement confrontés aux problèmes suivants :

- Nécessité de recompiler l'application à chaque modification d'un élément contenu dans l'application principale.
- Application principale lourde et longue à charger.
- Difficulté pour organiser convenablement et efficacement les assets et éléments d'une application.

Il est donc possible d'optimiser le flux de chargement des différents éléments et dépendances de l'application en utilisant l'attribut "**import**" dans tous les fichiers de configuration externes de la fabrique IoC chargés avec la classe **ECMAObjectLoader**.

Nous pouvons donc définir dans le chargeur de la fabrique **IoC** des ressources qui permettent en fonction de leur type de charger avec plus ou moins de complexité différents éléments externes indispensables pour alimenter dynamiquement l'application des éléments qui lui manquent.

Nous retrouvons donc dans le package principal andromeda.ioc le package **andromeda.ioc.io.*** qui contient des objets de types **ObjectResource**.

La classe **ObjectResource** est une classe simple qui définie des objets qui automatisent la création de modules indépendants ayant pour objectif de charger différents éléments externes dans l'application pendant l'initialisation de la fabrique et de lancer pour chacun de ces éléments des actions diverses comme un câblage automatique de l'objet avec une définition d'objet dans le conteneur **IoC**.

La classe **http://www.ekameleon.net/vegas/docs/andromeda/ioc/io/ObjectResource.html** implémente un design pattern de type "**Value Object**". Ce pattern permet d'assurer l'utilisation d'un objet simple contenant toutes les valeurs nécessaires pour définir et charger une ressource externe en fonction d'un type et parfois d'un identifiant défini par l'utilisateur dans le fichier de configuration de la fabrique. Les objets de types **ObjectResource** (qui héritent de cette classe) pourront être définis simplement en **eden** ou transférés via un protocol **AMF** par exemple avec **Flash Remoting** ou autre.

Les objets de type **ObjectResource** contiennent tous au minimum les attributs suivants :

description:String	Cet attribut facultatif permet simplement d'affecter une description de la ressource chargée, cette description peut être utilisée en cas de besoin pendant le chargement de cette ressource externe.
id:String	Identifiant du "value object" qui permet en cas de besoin de traiter convenablement la ressource externe une fois chargée (La classe **ObjectResource** implémente les interfaces andromeda.vo.IValueObject et vegas.core.Identifiable).
resource:String	Cet attribut (obligatoire) contient une chaine de caractère qui permet en général de définir l'uri du fichier externe que l'on souhaite charger. Parfois cet attribut prendra pour valeur une expression permettant de définir exactement la stratégie utilisée pour charger le fichier externe.
owner:*	Référence vers l'objet auquel appartient la resource chargée (en général référence vers la fabrique IoC ou le chargeur qui permet de configurer la fabrique). Cet attribut est utilisé en interne dans le moteur du chargeur de configuration de la fabrique.
title:String	Cet attribut facultatif permet de définir un titre que nous pourrons utiliser pendant le chargement de la ressource externe pour informer l'utilisateur de l'application.
type:String	Permet de définir le type de la resource externe et de définir la stratégie utilisée dans le chargeur pour initialiser chaque objet charger pour ce même type.

La classe **ObjectResource** contient également une méthode **create():CoreActionLoader** qui sera écrasée (**override**) dans les classes qui héritent de **ObjectResource** et qui contiendra précisément quel type de commande sera utilisée pour charger la ressource externe définie dans l'objet de type **ObjectResource** courant.

Cette méthode est utilisée en interne dans le chargeur du contexte de la fabrique **IoC** pour insérer dans une séquence tous les chargeurs spécialisés nécessaires pour charger les différentes ressources externes définies dans l'attribut "**imports**".

La classe **CoreActionLoader** est une classe basée sur le modèle d'action et de séquençage de **AndromedAS** (**package andromeda.process.***) qui permet de définir un ensemble d'action qui encapsule les classes **Loader**, **URLLoader**, etc. pour leurs permettre d'être injectées dans un séquenceur qui permettra de charger chaque ressources les unes après les autres.

I-C - Les différents types de ressources externes définies par défaut dans le chargeur ECMAObjectLoader.

Les objets définis dans **AndromedAS** qui héritent de la classe **ObjectResource** sont injectés dans une constante singleton définie dans le package **andromeda.ioc.io** avec l'objet **ObjectResourceBuilder**. Ce singleton est un modèle simple qui permet d'abonner des objets de type **ObjectResource** en fonction d'un type String unique (identifiant de la ressource à ne pas confondre avec l'attribut "**id**" des objets de type **ObjectResource**).

Ce modèle singleton permet au chargeur **ECMAObjectLoader** de récupérer dans une liste définie dans l'attribut "**imports**" tous ses objets génériques et en fonction de leur attribut "**type**" de créer et d'initialiser les objets de type **ObjectResource** correspondant.

Le singleton **ObjectResourceBuilder** contient les 3 méthodes suivantes :

addObjectResource(type:String , clazz:Class):Boolean	Cette méthode permet d'abonner une classe de type ObjectResource pour un type de resource donné.
get(o:Object):ObjectResource	Cette méthode est utilisée dans le chargeur de contexte externe pour créer et initialiser le nouvel objet de type ObjectResource en fonction d'un objet générique récupéré dans la liste des objets génériques de l'attribut "imports" défini dans un fichier de contexte externe. A noter qu'il faut absolument que l'objet générique contienne au moins un attribut "resource".
removeObjectResource(type:String):Boolean	Cette méthode permet de désabonner pour un type en particulier sa classe qui hérite de la classe ObjectResource (si elle existe)

Il est donc possible d'aller plus loin et de créer nos propres stratégies pour charger des ressources externes via le moteur de chargement du contexte du conteneur léger IoC. Il suffit pour cela de créer une classe qui hérite de la classe ObjectResource, d'écraser convenablement (override) la méthode create() de votre classe et ensuite de l'abonner dans le singleton ObjectResourceBuilder.

Par défaut la classe **ECMAObjectLoader** peut charger différentes ressources définies par défaut dans le chargeur de base :

```
// AndromedAS resources

ObjectResourceBuilder.addObjectResource( null                        , ContextResource    ) ;
ObjectResourceBuilder.addObjectResource( ObjectResourceType.CONTEXT  , ContextResource    ) ;
```

```
ObjectResourceBuilder.addObjectResource( ObjectResourceType.ASSEMBLY , AssemblyResource   ) ;

ObjectResourceBuilder.addObjectResource( ObjectResourceType.CONFIG   , ConfigResource     ) ;
ObjectResourceBuilder.addObjectResource( ObjectResourceType.I18N     , LocaleResource     ) ;

ObjectResourceBuilder.addObjectResource( ObjectResourceType.XML      , XMLResource        ) ;

// ASGard resources

ObjectResourceBuilder.addObjectResource( ObjectResourceType.STYLE    , StyleSheetResource ) ;
ObjectResourceBuilder.addObjectResource( ObjectResourceType.FONT     , FontResource       ) ;
```

Remarque : Comme vous pouvez le voir dans les déclarations ci-dessus, si l'utilisateur défini un objet générique sans type (null) dans la liste des "**imports**" du contexte externe, un objet de type **ContextResource** sera automatiquement utilisé pour charger un fichier au format eden qui contiendra un complément pour le contexte courant.

Voyons donc en détail les différentes classes de type **ObjectResource** disponibles et que nous pouvons utiliser par défaut dans une application utilisant la classe ECMAObjectLoader :

ContextResource dans le package andromeda.ioc.io avec le type "context" ou null.	Charge d'autres fichiers de configurations pour remplir le conteneur léger IoC.
AssemblyResource dans le package andromeda.ioc.io avec le type "assembly"	Charge dynamiquement des classes ou des assets nécessaires pour alimenter le contenu de l'application.
ConfigResource dans le package andromeda.config avec le type "config"	Charge un fichier de configuration au format eden (ou autre) pour initialiser ou compléter le contenu de l'objet "config" de configuration de la fabrique.
LocaleResource dans le package andromeda.system avec le type "i18n"	Charge un fichier de localisation en fonction d'un identifiant de langue ("en", "fr", etc.) et initialiser l'objet locale correspondant.
StyleSheetResource dans le package asgard.text avec le type "style"	Charge une feuille de style CSS externe et de l'affecter dans la fabrique par référence dans une définition d'objet facilement utilisable dans les autres définitions d'objets
FontResource dans le package asgard.text avec le type "font"	Charge des fichiers au format swf externes qui contiennent des polices de caractères "embed" et de définir les classes des polices utilisées dans l'application par la suite.
XMLResource dans le package andromeda.ioc.io avec le type "xml"	Charge un fichier texte externe contenant un DOM XML. Une fois ce XML chargé il est automatiquement lié à une définition d'objet dans la fabrique.

Voyons maintenant un exemple complet de définitions de différents types de ressources chargées dans un contexte externe de configuration IoC :

```
imports =
[

    // assembly

    { resource : "library/dll.swf" , type: "assembly" }
    ,
    {
        id          : "assembly_picture" ,
        resource    : "library/picture.jpg" ,
        type        : "assembly" ,
```

```
        definition : { type : "flash.display.Bitmap" }
    }
    ,

    // config

    { resource : "config" , type : "config"  , path : "config/" , suffix : ".eden"  ,
 verbose : true } ,

    // fonts

    { resource : "fonts/fonts.swf" , type : "font" , fonts : [ "ArialBlack", "MyriadPro" ] } ,

    // i18n (localization)

    { resource : "fr" , type : "i18n" , prefix : "localize_" , path : "locale/" , suffix : ".eden" ,
 verbose : true } ,

    // style

    { resource : "style/style.css" , type : "style" , id : "style_sheet" } ,

    // context

    { resource : "user.eden"  , type : "context" } ,
    { resource : "view.eden" } ,

    // xml

    { resource : "data/data.xml" , type : "xml" , id : "data"  , path : "data/" }

 ] ;
```

Nous allons voir en détail dans les prochains article la méthodologie établie pour utiliser chaque type de ressource définie dans l'exemple ci-dessus.

Cette liste de ressources peut être définie selon les besoins dans tous les fichiers de contexte externe chargés dans un même processus d'initialisation d'une fabrique **IoC** avec son chargeur de type **ECMAObjectLoader**.

Le chargeur **IoC** va donc redéfinir l'ensemble des **objets génériques** définis ci-dessus en vérifiant que chaque **objet générique** :

- contient au moins un attribute "resource" valide (type String).
- possède un attribut "type" compatible avec un type enregistré dans le modèle **ObjectResourceBuilder**, dans le cas contraire il va créer par défaut une ressource de type "**context**" avec la classe **ContextResource**.

Ensuite il va générer et initialiser chaque ressource avec un séquenceur interne.

A noter que les **définitions d'objet** définies dans certaines ressources (dans les ressources de types "**style**" ou "**assembly**" par exemple) seront automatiquement injectés dans le conteneur léger pendant l'initialisation de chacune d'elles et seront utilisables par référence dans la fabrique dès que le chargement des ressources sera terminé.

II - Les ressources de type "context"