VEGAS - Design Pattern d'inversion de contrôle : Configuration d'une fabrique IoC

par Marc Alcaraz (Mon site)

Date de publication : 11/09/2009

Dernière mise à jour :

Dans cet article, comme son titre l'indique, nous allons aborder dans cet article la configuration des fabriques **IoC**.

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**

I - Introduction

Les fabriques **IoC** dans **AndromedAS** sont configurables. Il est possible de définir les paramètres généraux de configuration d'une fabrique en alimentant le contenu de sa propriété "**config**".

La propriété "**config**" est un objet de type **andromeda.ioc.factory.ObjectConfig**. Voici la liste de ses attributs disponibles dans cet objet de configuration :

config

Défini un objet servant de configuration globale dans la fabrique ou l'application utilisant cette fabrique.

defaultInitMethod

Défini le nom de la méthode par défaut à utiliser dans toutes définitions d'objet au moment de l'initialisation d'un objet généré avec l'une d'elles.

defaultDestroyMethod

Défini le nom de la méthode par défaut à utiliser dans toutes définitions d'objet au moment de supprimer un objet singleton généré avec l'une d'elles.

identify

Défini globalement la politique d'identification automatique des objet qui implémentent l'interface Identifiable générés dans la fabrique IoC.

locale

Défini un objet localisé servant de base pour alimenter les définitions d'objet de la fabrique selon les besoins avec des valeurs localisées.

lock

Défini globalement la politique du mode de sécurité implémenté par tous les objets ILockable générés via la fabrique **IoC**.

root

Défini la référence de la scène principale de l'application liée à la fabrique IoC, cette valeur permet d'utiliser dans les attributs "ref" des définitions d'objet l'expression "#root"

throwError

Défini ou désactive la diffusion des erreurs pendant l'initialisation de la fabrique. Par défaut cet attribut est true.

typeAlias

Défini les alias utilisés au moment de filtrer les types dans les définitions d'objet (si la propriété typePolicy a pour valeur "alias" ou "all").

typeExpression

Défini les expressions de formatage utilisées pour filtrer les types dans les définitions d'objet (si la propriété typePolicy a pour valeur "expression" ou "all").

typePolicy

Défini la politique de filtrage du type défini dans les définitions d'objet du conteneur léger au moment d'instancier un nouvel objet pour un identifiant donné.

useLogger

Défini le mode de diffusion des messages d'information, warning, erreurs, etc. de la fabrique entre un simple trace() ou le modèle de log défini dans VEGAS.

Voici un exemple d'utilisation **manuelle** des attributs de configuration d'une fabrique **IoC** :

```
import andromeda.ioc.core.TypePolicy ;

import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

// Sets the root reference use in the factory to target the "stage" and the "root"

conf.root  = this ;
conf.stage = stage ; // optional if the root property is defined.

// config and localization configuration

var conf:ObjectConfig         = factory.config ;

conf.config                   = { color:0xFF0000 , alpha:1 } ;
conf.locale                   = { message1:"hello world" , message2:"hi city" } ;

// life cycle attributes

conf.defaultInitMethod        = "init" ;
conf.defaultDestroyMethod     = "destroy" ;
conf.identify                 = true ;
conf.lock                     = true ;

// debug attributes

conf.throwError               = true ;
conf.useLogger                = false ;

// type evaluator configuration

conf.typePolicy               =
 TypePolicy.ALL ; // TypePolicy.NONE, TypePolicy.ALIAS, TypePolicy.EXPRESSION
conf.typeAliases              = [ { alias:"HashMap" , type:"system.data.maps.HashMap" } ] ;
conf.typeExpression           =
[
    { name:"maps"     , value:"system.data.maps" } ,
    { name:"ArrayMap" , value:"{maps}.ArrayMap"  }
] ;
```

Dans le fichier externe principal de configuration d'un conteneur **IoC** nous pouvons définir, dans le premier contexte externe chargé, l'attribut "**configuration**". A noter que cet objet ne peut être utilisé que dans le premier fichier de configuration chargé avec une instance de la classe **ECMAObjectLoader**.

Voyons maintenant, en se basant sur l'exemple ci-dessus, comment définir la configuration de la fabrique IoC dans un fichier de configuration externe au format **eden** :

```
configuration =
{

    // config and localization configuration

    config :
    {
        color:0xFF0000 , alpha:1
    }
    ,
    locale :
    {
        { message1 : "hello world" , message2 : "hi city" }
    }
    ,

    // life cycle attributes

    defaultInitMethod    : "initialize" ,
    defaultDestroyMethod : "destroy"    ,
    identify             : true         ,
    lock                 : true         ,

    // type evaluator configuration

    typePolicy  : "all" ,
    typeAliases :
    [
        { alias : "TextField" , type : "flash.text.TextField" } ,
        { alias : "User"      , type : "test.User"            }
    ]
    ,
    typeExpression :
    [
        { name : "display" , value : "asgard.display" }
    ]
    ,

    // debug attributes

    throwError : true  ,
    useLogger  : false

} ;
```

*A noter que l'attribut **root** peut être défini dans le contexte externe de configuration de la fabrique mais il est préférable de l'initialiser directement dans le code source de l'application juste avant de lancer le chargement de la configuration externe.*

II - L'attribut "config"

L'attribut "**config**" est un attribut spécial défini dans la configuration de la fabrique **IoC**. Cette fonctionnalité n'est pas une fonctionnalité classique que l'on peut retrouver dans d'autres frameworks qui implémentent le design pattern d'**Inversion de contrôle**. Elle s'inscrit dans la nécessité de pouvoir accéder facilement dans les **définitions d'objets** aux valeurs définies dans un objet de configuration simple contenant toutes les variables importantes d'une application.

La fabrique **IoC** basée sur la classe **ObjectFactory** contient une configuration interne via une instance de la classe **ObjectConfig** et cette instance contient un attribut dynamique nommé "**config**" qui peut être rempli avec toutes sortes d'attributs qui pourront être réutilisés dans la fabrique et ses **définitions d'objets** le moment voulu. Les attributs de cet objet **config** servent en quelques sorte de variables dynamiques que l'on peut définir dans l'application dans le fichier principal de configuration externe du conteneur léger.

Dans les **définitions d'objets**, il est possible dans les attributs "**properties**" ou "**arguments**" d'utiliser à la place des attributs "**value**" ou "**ref**" l'attribut "**config**" qui récupère une valeur dans l'objet "**config**" de la fabrique.

L'attribut "**config**" de type **String** utilise la classe **andromeda.ioc.evaluators.ConfigEvaluator** pour évaluer correctement l'expression définie dans une **définition d'objet** pour renvoyer la valeur de la variable définie dans la configuration du conteneur léger.

Il est possible d'utiliser indépendamment de la fabrique **IoC** cette classe (voir chapitre B-4-2-1 - La classe ConfigEvaluator).

Voyons maintenant comment déclarer l'attribut "**config**" dans un fichier de configuration externe au format **eden** :

```
configuration =
{
    config :
    {
        message : "hello world" ,
        menu    :
        {
            title : "my title" ,
            count : 10 ,
            data  : [ "item1" , "item2", "item3" ]
        }
    }
} ;

objects =
[
     {
         id         : "my_message" ,
         type       : "flash.text.TextField" ,
         properties :
         [
             { name : "text" , config : "message" }
         ]
     }
     ,
     {
         id         : "my_menu_title" ,
         type       : "flash.text.TextField" ,
         properties :
         [
             { name : "text" , config : "menu.title" }
         ]
     }
] ;
```

Dans l'exemple ci-dessus, les attributs "**properties**" et "**arguments**" utilisent l'attribut "**config**" pour récupérer une valeur définie dans l'objet de configuration de la fabrique. L'attribut "**config**" est défini par une expression (**String**) avec une notation pointée pour récupérer en profondeur dans les attributs de l'objet la valeur désirée.

A noter que l'attribut "**config**" de la classe **ObjectConfig** est une propriété virtuelle (*get/set*) un peu spéciale définie avec les spécifications suivantes :

- L'attribut en mode "**get**" renvoit simplement la référence de l'objet dynamique définie en interne dans l'instance de type **ObjectConfig**.
- L'attribut en mode "**set**" prend pour valeur un objet "dynamique" qui sera copié membre à membre dans l'objet de configuration interne

Il est donc impossible de vider l'objet de configuration en tapant simplement :

```
import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;
var conf:ObjectConfig = factory.config ;

conf.config = { prop1 : "value1" , prop2 : "value2" } ;
trace( conf.config.prop1 ) ; // value1

conf.config  = {} ;
trace( conf.config.prop1 ) ; // value1
```

Pour vider complètement l'objet de configuration interne il faudra utiliser la méthode '**resetConfigTarget()**' de la classe **ObjectConfig** :

```
import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;
var conf:ObjectConfig = factory.config ;

conf.config = { prop1 : "value1" , prop2 : "value2" } ;
trace( conf.config.prop1 ) ; // value1

conf.resetConfigTarget() ;
trace( conf.config.prop1 ) ; // undefined
```

A noter qu'il est possible de redéfinir la référence interne de l'objet de configuration utilisée par l'instance de type **ObjectConfig** en utilisant la méthode **setConfigTarget(o:Object)**. Cette méthode prend en paramètre un objet "*dynamic*" qui servira de référence pour l'objet de configuration de la fabrique **IoC**. A noter que si l'objet passé en paramètre est '*null*' ou non dynamique l'instance de type **ObjectConfig** est initialisé avec un simple objet de type **Object**.

Si la fabrique IoC est initialisée avec un chargeur de type **ECMAObjectLoader** alors l'attribut **config** de la configuration de la fabrique fait référence au singleton de type **andromeda.config.Config** défini dans une application avec l'instruction

```
ECMAObjectFactory.getInstance().config.setConfigTarget( Config.getInstance() ) ;
```

Ce singleton de type **Config** est lié dans **AndromedAS** à un moteur de configuration automatisé basé sur un chargement de fichiers **eden** ou **json** (possible d'étendre avec tout autre format ou protocole). Je ne vais pas entrer ici dans les détails de cette implémentation mais il est intéressant de pouvoir faire le pont entre les définitions d'objets de la **fabrique IoC** et l'**objet de configuration global** d'une application.

Cette spécificité permet de lier l'objet de configuration au chargement de plusieurs ressources externes de type "**config**" qui permettent à tout moment dans les fichiers de configuration de la fabrique de charger des fichiers généralement au format eden beaucoup plus simple qui permettent de remplir progressivement l'objet de configuration global de l'application (D.2-3 à propose des ressources externes et du type de ressource "**config**").

III - Les attributs "defaultInitMethod" et "defaultDestroyMethod".

Cet deux attributs permettent de définir le nom d'une méthode d'initialisation ou d'une méthode de destruction à appliquer (si elles sont définies pour l'objet) sur tous les objets générés par la fabrique **IoC**.

Il est possible de définir directement via les attributs **init** et **destroy** ces noms de méthodes sur une **définition d'objet**.

Il faut noter que les attributs **init** et **destroy** d'une définition d'objet sont toujours prioritaires sur les noms de méthodes définis dans la configuration de la fabrique.

Ces méthodes doivent être définies avec aucun argument.

```
import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;
var conf:ObjectConfig         = factory.config

conf.defaultInitMethod        = "init" ;
conf.defaultDestroyMethod     = "destroy" ;

trace( conf ) ; // [ObjectConfig defaultDestroyMethod:destroy defaultInitMethod:init identify:false lock:false]
```

Vous pouvez consulter les chapitres B-2.1 et B-2.2 au sujet des attributs **init** et **destroy** d'une définition d'objet pour en savoir plus sur l'utilisation de ces 2 attributs liés au cycle de vie des objets défini par une fabrique **IoC** dans l'application.

IV - L'attribut "identify"

Dans l'objet de configuration de la fabrique **IoC** nous pouvons définir un attribut booléen général "**identify**" (comme dans les définitions d'objet).

Cet attribut possède une valeur par défaut "**false**" si elle n'est pas définie dans l'objet de configuration. Cet attribut indique au conteneur **IoC** si il doit vérifier au moment de créer un objet si l'objet est un "**singleton**" et surtout s'il implémente l'interface **http://www.ekameleon.net/vegas/docs/system/data/Identifiable.html**.

```
package system.data
{
    public interface Identifiable
    {
        function get id():* ;

        function set id( id:* ):void ;
    }
}
```

Si l'attribut a une valeur "**true**" et que les 2 critères ci-dessus sont bien respectés (singleton + Identifiable) alors tous les singletons dans ce cas auront leur propriété '**id**' (définie dans l'interface **Identifiable**) initialisée avec la valeur de la propriété "**id**" de la **définition d'objet** courante.

```
configuration =
{
    identify : true
} ;

objects =
[
    {
        id        : "my_sprite" ,
        type      : "asgard.display.CoreSprite" ,
        singleton : true // must be a singleton
    }
] ;
```

L'exemple précédent définit un objet de type **asgard.display.CoreSprite**. Cette classe spécifique à l'extension **ASGard** de **VEGAS** hérite de la classe **flash.display.Sprite** et contient quelques fonctionnalités communes à tous les objets graphiques de **ASGard**. Cette classe **CoreSprite** implémente donc l'interface **http://www.ekameleon.net/vegas/docs/system/data/Identifiable.html**.

Voyons maintenant avec un exemple **AS3** simple ce qu'il se passe exactement lorsque l'attribut "**identify**" est défini avec une valeur "**true**".

```
import andromeda.ioc.factory.ECMAObjectFactory ;
import asgard.display.CoreSprite ;

var mySprite:CoreSprite = ECMAObjectFactory.getInstance().getObject("my_sprite") as CoreSprite ;

trace( mySprite.id ) ; // my_sprite
```

Le nouveau objet de type **CoreSprite** est créé par la fabrique **IoC** et sa propriété **id** possède automatiquement la valeur de l'identifiant de la **définition d'objet** correspondante.

*Si, dans la définition de l'objet l'attribut, "**identify**" est défini avec une valeur "**false**" et que dans la configuration de la fabrique l'attribut "**identify**" global possède une valeur "**true**" alors la propriété id de l'objet **Identifiable** ne sera pas automatiquement initialisé.*

L'attribut "**identify**" d'une **définition d'objet** est prioritaire sur la propriété "**identify**" de l'objet de configuration de la fabrique.

V - L'attribut "locale"

L'attribut "**locale**" fonctionne à peu prêt comme l'attribut "**config**" défini précédemment. Cet attribut spécial est défini dans la configuration de la fabrique **IoC** et permet d'accéder facilement dans les **définitions d'objets** aux valeurs définies dans un objet de configuration localisé simple contenant toutes des variables importantes d'une application en fonction d'une langue en particulier.

La fabrique **IoC** basée sur la classe **ObjectFactory** contient une configuration interne via une instance de la classe **ObjectConfig** et cette instance contient par défaut un **objet générique** dynamique nommé "**locale**" qui peut être rempli avec toutes sortes d'attributs qui pourront être réutilisés dans la fabrique et ses **définitions d'objets** le moment voulu dans les **définitions d'objet** du **conteneur IoC**.

Dans les **définitions d'objets** du conteneur **IoC**, il est donc possible dans les attributs "**properties**" ou "**arguments**", d'utiliser en plus des attributs "**value**" ou "**ref**" l'attribut "**locale**" qui pourra chercher la valeur voulue dans l'objet "**locale**" de la fabrique. L'attribut "**locale**" de type **String** utilise la classe **andromeda.ioc.evaluators.LocaleEvaluator** pour évaluer correctement la valeur de la variable définie dans la configuration du conteneur léger (Il est possible d'utiliser indépendamment de la fabrique **IoC** cette classe - voir chapitre B-4.2.2 - **La classe LocaleEvaluator**).

Voyons maintenant un exemple simple pour illustrer comment utiliser l'attribut "**locale**" dans un fichier de configuration externe au format **eden** :

```
configuration =
{
    locale :
    {
        en :
        {
            message : "hello world" ,
            menu    :
            {
                title : "my title"
            }
        }
        ,
        fr :
        {
            message : "bonjour le monde" ,
```

```
                menu    :
                {
                    title : "mon titre"
                }
            }
        }
    } ;

    objects =
    [
        {
            id          : "my_message" ,
            type        : "flash.text.TextField" ,
            properties :
            [
                { name : "text" , locale : "en.message" }
            ]
        }
        ,
        {
            id          : "my_menu_title" ,
            type        : "flash.text.TextField" ,
            properties :
            [
                { name : "text" , locale : "fr.title" }
            ]
        }
    ] ;
```

Il est donc possible d'utiliser dans les attributs "**properties**" et "**arguments**" des définitions d'objets l'attribut "**locale**" pour récupérer une valeur définie dans l'objet de localisation de base défini dans la fabrique. L'attribut "**locale**" est défini par une expression (**String**) avec une notation pointée pour récupérer en profondeur dans les attributs de l'objet la valeur désirée.

A noter que l'attribut "**locale**" de la classe **ObjectConfig** est une propriété virtuelle (*get/set*) un peu spéciale définie avec les spécifications suivantes :

- L'attribut en mode "**get**" renvoit simplement la référence de l'objet dynamique définie en interne dans l'instance de type **ObjectConfig**.
- L'attribut en mode "**set**" prend pour valeur un objet dynamique qui sera copié membre à membre dans l'objet de configuration interne

Il est donc impossible de vider l'objet de configuration en tapant simplement :

```
import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;
var conf:ObjectConfig = factory.config ;

conf.locale = { text1: "value1" , text2: "value2" } ;
trace( conf.locale.text1 ) ; // value1

conf.config  = {} ;
trace( conf.locale.text1 ) ; // value1
```

A noter qu'il est possible de redéfinir la référence interne de l'objet de configuration utilisé par l'instance de type **ObjectConfig** en utilisant la méthode **setLocaleTarget(o:Object)**. Cette méthode prend en paramètre un objet "*dynamic*" qui servira de référence pour l'objet de configuration de la fabrique **IoC**. A noter que si l'objet passé en paramètre est '*null*' ou non dynamique l'instance de type **ObjectConfig** est initialisé avec un simple objet de type **Object**.

```
import andromeda.ioc.factory.ECMAObjectFactory ;
```

```
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;
var conf:ObjectConfig = factory.config ;

conf.locale = { prop1 : "value1" , prop2 : "value2" } ;
trace( conf.locale.prop1 ) ; // value1

conf.setLocaleTarget( null ) ;
trace( conf.locale.prop1 ) ; // undefined
```

A noter que dans une utilisation avancée d'une fabrique **IoC** avec un chargeur de contexte externe, l'objet de localisation défini dans la configuration de la fabrique est lié au chargement de plusieurs ressources externes de type "**i18n**" qui permettent à tout moment dans les fichiers de configuration de la fabrique de charger des fichiers généralement au format eden beaucoup plus simple qui permettent de remplir progressivement l'objet de configuration global de l'application (voir chapitres C.2-3 et D.2-4 à propose des ressources externes et du type de ressource "**i18n**").

De plus l'objet locale est synchronisé avec le moteur de localisation de AndromedAS défini par le package **andromeda.i18n** et surtout la classe **Localization**. Il est donc possible de modifier à tout moment la langue courante d'une application et de switcher le contenu courant de la référence "**locale**" vers l'objet contenant tous les éléments (String, url, etc..) localisés correspondant à la langue choisie.

VI - L'attribut "lock"

Dans l'objet de configuration de la fabrique **IoC** nous pouvons définir un attribut booléen général "**lock**" (comme dans les **définitions d'objet** - voir chapitre B.2-4). Cet attribut possède une valeur par défaut "**false**" si elle n'est pas définie dans l'objet de configuration. Cet attribut charge le conteneur **IoC** de vérifier, au moment de créer un objet, si ce dernier implémente ou pas l'interface **http://www.ekameleon.net/vegas/docs/system/process/Lockable.html**.

Si l'attribut a une valeur "**true**" et que l'objet implémente l'interface **ILockable** alors tous ces objets verront leur méthode **lock()** invoquée avant l'initialisation de ses propriétés et l'invocation de certaines de ses méthodes toutes définies dans la définition de l'objet nouvellement créé, une fois le processus d'initialisation terminé la méthode **unlock()** de l'objet sera invoquée juste avant l'invocation d'une éventuelle dernière méthode d'initialisation définie dans la définition d'objet avec l'attribut "**init**".

VII - Les attributs "root" et "stage"

L'attribut "**root**" sera surtout utilisé dans le code source de l'application avant d'exécuter l'initialisation de la fabrique avec une configuration externe.

Exemple 1 : Définir la référence directement dans l'objet config de la fabrique

```
import andromeda.ioc.factory.ECMAObjectFactory ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

factory.config.root = this ; // this == root of the application
```

Exemple 2 : Définir la référence avec la référence root d'une instance de la classe ECMAObjectLoader.

```
import asgard.net.ECMAObjectLoader ;

var loader:ECMAObjectLoader = new ECMAObjectLoader( "application.eden" , "context/" ) ;

loader.root = this ; // to use the "#root" expression in the 'ref' attribute.
```

Il est donc possible maintenant d'utiliser l'expression magique "**#root**" dans tous les attributs '**ref**' ou dans les attributs **factoryReference** (voir chapitre B-1-8-3) des **définitions d'objet** de la fabrique **IoC**, exemple :

```
objects =
[
    {
        id               : "root" ,
        type             : "flash.display.MovieClip"  ,
        factoryReference : "#root" ,
        singleton        : true ,
        properties       :
        [
            { name : "addChild" , arguments  : [ { ref:"my_field" } ]}
        ]
    }
] ;
```

Il est également possible d'utiliser l'expression magique "**#stage**" dans tous les attributs '**ref**' ou dans les attributs **factoryReference** (voir chapitre B-3-3) des **définitions d'objet** de la fabrique **IoC** :

```
import andromeda.ioc.factory.ECMAObjectFactory ;

var context:Object =
[
     {
        id               : "stage" ,
        type             : "flash.display.Stage" ,
        factoryReference : "#stage" ,
        singleton        : true     ,
        properties       :
        [
            { name:"align"     , value:"tl"      } ,
            { name:"scaleMode" , value:"noScale" }
        ]
    }
] ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

factory.config.root = this ; // important to use the magic #root or #stage references

factory.create( context ) ;
```

Si la propriété **root** de l'objet de configuration cible bien la référence du **DisplayObject** principal de l'application alors sa propriété "**stage**" ne doit pas être **null**. La propriété "**root**" peut finalement être définie avec n'importe quel **DisplayObject** si il est attaché dans la "display list" de l'application.

Il est tout de même préférable d'utiliser cette propriété en ciblant la vraie référence principale de l'application (dans la classe principale de l'application).

Il est donc très facile pour la fabrique de renvoyer si elle existe la référence du **Stage** de l'application en ciblant celle ci avec la propriété "**stage**" du **DisplayObject** principal. Pour éviter toute ambiguité il est également possible et vivement conseillé d'utiliser l'attribut "**stage**" de la configuration qui prendra dans tous les cas le dessus sur la propriété "**root**" pour déterminer la référence du **Stage** avec la référence magique **#stage**.

```
import andromeda.ioc.factory.ECMAObjectFactory ;

var context:Object =
[
     {
        id               : "stage" ,
        type             : "flash.display.Stage" ,
        factoryReference : "#stage" ,
        singleton        : true     ,
        properties       :
        [
```

```
            { name:"align"     , value:"tl"      } ,
            { name:"scaleMode" , value:"noScale" }
        ]
    }
] ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

factory.config.stage = stage ; // important to use the magic #stage reference

factory.create( context ) ;
```

VIII - Les attributs de configuration des types dans les définitions d'objets

L'attribut "**type**" dans les définitions d'objets du conteneur **IoC** peuvent être filtrés et formatés en utilisant des outils prédéfinis dans la fabrique **IoC**. Il est donc possible de gérer au plus fin les niveaux de filtrages de cet attribut au moment d'instancier un nouvel objet dans la fabrique.

Pour activer et définir le filtrage de l'attribut "**type**" il faut définir correctement certains attributs dans l'objet de configuration de la fabrique (**ObjectConfig**). Voyons de plus prêt les différents attributs disponibles dans la configuration du conteneur **IoC**.

- **typePolicy** : Permet de d'activer ou désactiver le filtrage et de choisir le niveau de filtrage
- **typeAliases** : Permet d'utiliser des alias pour simplifier l'écriture des types dans les définitions d'objet d'une fabrique.
- **typeExpression** : Permet de formater les types dans les définitions d'objet en se servant d'un dictionnaire d'expressions spécifiques aux types définis dans le conteneur **IoC**.

VIII-A - L'attribut "typePolicy"

L'attribut "**typePolicy**" de la configuration de la fabrique **IoC** permet de définir la politique du filtrage effectué sur les attributs "**type**" dans les définitions d'objet au moment de créer un objet avec la fabrique.

- "**none**" : Aucun filtre sur les types dans les définitions d'objets.
- "**alias**" : Filtrage des types avec des alias contenus dans une Map (**TypeAlias**)
- "**all**" : Active tous les filtrages sur l'attribut type dans les définitions d'objets au moment de la création des objets avec la fabrique.
- "**expression**" : Filtrages des titres avec un formatage via un dictionnaire d'expressions (**TypeExpression**)

A noter qu'il existe une classe d'énumération des différentes valeurs que peut prendre l'attribut '**typePolicy**' avec la classe **andromeda.ioc.core.TypePolicy** avec les constantes :

- **TypePolicy.ALIAS**
- **TypePolicy.ALL**
- **TypePolicy.EXPRESSION**
- **TypePolicy.NONE**

Si cet attribut possède la valeur "**alias**" ou "**expression**" ou "**all**", à chaque invocation de la méthode **getObject()**, la fabrique va utiliser son objet de configuration pour formater et évaluer la chaîne de caractère du type défini dans la définition.

L'utilisation de ces filtres permet de simplifier et d'optimiser énormément le contenu et la taille des fichiers de configuration externes de la fabrique **IoC** en créant des raccourcis et des expressions simples pour cibler les types d'objets. Par contre, ces filtres ralentissent un peu la création des objets, il faudra donc bien définir selon la nature des projets l'utilisation ou non de ces filtrages.

Si le type "**typePolicy**" n'est pas défini dans l'objet de configuration alors il aura pour valeur par défaut "**none**" et donc aucun filtrage ne sera effectué sur le type de la définition d'objet.

VIII-B - L'attribut "typeAliases"

Dans une **définition d'objet** cet attribut contient une liste (Array) d'**objets génériques** définis avec les attributs "**alias**" et "**type**". Ces objets permettent de remplir l'attribut "**typeAliases**" de l'objet de configuration de la fabrique.

```
configuration =
{
    typePolicy   : "alias" ,
    typeAliases  :
    [
        { alias:"Sprite"    , type:"flash.display.Sprite" } ,
        { alias:"TextField" , type:"flash.text.TextField" }
    ]
} ;

objects :
[
 { id:"my_sprite" , type:"Sprite" } ,
    { id:"my_field"  , type:"TextField" }
];
```

Voyons maintenant comment remplir la configuration directement avec du script ActionScript classique :

```
import andromeda.ioc.core.TypePolicy ;

import andromeda.ioc.factory.ECMAObjectFactory ;
import andromeda.ioc.factory.ObjectConfig ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

var conf:ObjectConfig         = factory.config

conf.typePolicy               = TypePolicy.ALIAS ;
conf.typeAliases              = [ { alias:"CoreObject" , type:"vegas.core.CoreObject" } ] ;
```

VIII-C - L'attribut "typeExpression"

Dans une **définition d'objet** l'attribut "**typeExpression**" contient une liste (Array) d'objets génériques définis avec les attributs "**name**" et "**value**".

Ces objets génériques permettent de remplir la collection "**typeExpression**" définie dans la configuration du conteneur et de créer des objets de type **TypeExpression** qui seront utilisés ensuite pour formater correctement la chaine de caractère définissant le type de l'objet que l'on souhaite créer avec la fabrique.

```
configuration =
{
    typePolicy    : "expression" ,
    typeExpression :
    [
        { name:"display" , value:"flash.display" } ,
        { name:"text" , value:"flash.text"  } ,
       { name:"data" , value:"vegas.data"  } ,
 { name:"HashMap" , value:"{data}.map.HashMap" }
    ]
} ;

objects :
```

```
[
   { id:"my_sprite" , type:"{display}.Sprite" } , // type:"flash.display.Sprite"
   { id:"my_field"  , type:"{text}.TextField" } , // type:"flash.text.TextField"
   { id:"my_map"    , type:"{HashMap}"    }  // type:"vegas.data.map.HashMap"
] ;
```

Il est également possible de remplir la configuration des expressions permettant de formater les types des définition d'objets directement avec dans le code de l'application :

```
import andromeda.ioc.core.TypePolicy ;


import andromeda.ioc.factory.ECMAObjectFactory ;

import andromeda.ioc.factory.ObjectConfig ;


var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

var conf:ObjectConfig         = factory.config ;


conf.typePolicy = TypePolicy.EXPRESSION ;


conf.typeExpression  =
[
    { name:"display"  , value:"flash.display"      } ,
    { name:"text"     , value:"flash.text"         } ,

    { name:"data"     , value:"vegas.data"         } ,

    { name:"HashMap"  , value:"{data}.map.HashMap" }
] ;
```

IX - Gestion des erreurs et des messages avec les attributs "throwError" et "userLogger"

La configuration interne d'une fabrique **IoC** contient 2 propriétés qui permettent de controler le niveau de debug et de la gestion des erreurs en interne pendant l'initialisation du conteneur léger.

Ces 2 propriétés servent avant tout pour le debug de vos applications, il suffit de jouer un peu avec les valeurs "**true**" ou "**false**" de celles-ci pour voir rapidement leur champ d'action.

```
import andromeda.ioc.factory.ObjectConfig ;
import andromeda.ioc.factory.ECMAObjectFactory ;

import system.logging.LoggerLevel ;
import system.logging.targets.TraceTarget ;

// setup writer

var traceTarget:TraceTarget = new TraceTarget() ;

traceTarget.filters      = ["*"] ;
traceTarget.includeLines = true ;
traceTarget.includeTime  = true ;
traceTarget.level        = LoggerLevel.ALL ;

// setup IoC context

var context:Object =
[
    {
        id        : "test" ,
        type      : "String" ,
```

```
        arguments :
        [
            { value : "hello world" }
        ]
    }
    ,
    {
        id        : "o" ,
        type      : "Object" ,
        arguments :
        [
            { value : { label:"hi world" } }
        ]
    }
    ,
    {
        id              : "test3" ,
        type            : "String" ,
        factoryReference : "o.labels"
    }
] ;

var factory:ECMAObjectFactory = ECMAObjectFactory.getInstance() ;

factory.create( context ) ;

trace("------- test a valid id in the factory") ;

trace( factory.getObject("test") ) ;

trace("-------  use trace when a warning log message is invoked in the factory") ;

factory.config.useLogger = false ;

trace( factory.getObject("test1") ) ;

trace("------- use the ILogger object defines by default in the factory") ;

factory.config.useLogger = true ; // default value

trace( factory.getObject("test2") ) ;

trace("------- Change evaluation errors with the throwError flag") ;

factory.config.throwError = true ; // enabled all errors
trace( factory.getObject("test3") ) ;

factory.config.throwError = false ; // disabled all errors
trace( factory.getObject("test3") ) ;
```