

Bean Script Framework

Philippe PRADOS

pp@philippe.prados.name



*Préservez l'environnement,
n'imprimez pas ce document*

Avant propos

Ce document résume les fonctionnalités de Bean Script Framework.

Nous avons regardé dans un précédent article la librairie Rhino permettant d'intégrer du JavaScript à un programme Java. D'autres langages peuvent être associés à Java. Par exemple, il existe une implémentation de Python ou un interpréteur TCL pour Java.

Pour normaliser l'invocation de ces différents langages, IBM a proposé une API normalisée permettant d'invoquer tous les langages de script. La librairie Bean Script Framework (BSF) possède différents connecteurs pour invoquer de nombreux langages. De nouveaux connecteurs peuvent être ajoutés pour s'interfacer avec de nouveaux langages.

Avec la librairie BSF, un programme java peut invoquer un script en faisant abstraction du langage utilisé. Ainsi, les rédacteurs du script peuvent choisir le langage qu'ils préfèrent.

Cette librairie a été initialement rédigée pour permettre l'utilisation de différents langages dans les pages JSP. Par défaut, les scripts des pages JSP sont rédigés en Java. IBM avec WebSphere permet d'indiquer un autre langage dans un marqueur `<%@ page %>`. IBM a rédigé à cette occasion la librairie BSF et l'a proposée à la communauté du libre.

Cette librairie est maintenant utilisée dans le moteur Xalan afin de permettre l'invocation de script dans un filtre XSL.

Pour utiliser cette librairie, il faut la télécharger ici : <http://oss.software.ibm.com/developerworks/projects/bsf>. Il faut également télécharger les librairies java pour les différents langages à utiliser. BSF accepte par défaut les langages du Tableau 1.

javascript	http://www.mozilla.org/rhino/
netrexx	http://www2.hursley.ibm.com/netrexx/
jacl	http://www.scriptics.com/software/
jpython	http://www.jpython.org/
java	http://java.sun.com/
vbscript	http://msdn.microsoft.com/scripting/
jscript	http://msdn.microsoft.com/scripting/
perlscript	http://www.activestate.com

Tableau 1

Pour invoquer un script, il faut créer une instance `BSFManager`, puis obtenir un moteur pour un langage particulier.

```
BSFManager manager=new BSFManager();
BSFEngine rhino=manager.loadScriptEngine("javascript");
```

Nous pouvons alors évaluer ou exécuter un morceau de script.

```
Object result=rhino.eval("test",0,0,"3+0.14");
rhino.call(null,"coucou",null);
rhino.exec("test",0,0,"coucou()");
```

Nous pouvons simplifier cela en demandant directement au manager d'exécuter un script pour un langage donné.

```
manager.exec("javascript","test",0,0,"coucou()");
```

Il est également possible de demander la compilation d'un script afin de pouvoir l'utiliser plusieurs fois avec plus d'efficacité.

```
CodeBuffer cb=new CodeBuffer();
rhino.compileScript("test",0,0,"coucou()",cb);
String res=cb.toString();
```

Si le langage ne peut pas être compilé en p-code Java, l'instance `CodeBuffer` invoque l'interpréteur du langage.

BSF offre une interface en ligne de commande pour invoquer directement un script avec le langage de son choix.

```
java com.ibm.bsf.Main -language javascript -in test.js
```

Pour qu'un script puisse manipuler l'application l'invoquant, il est nécessaire d'initialiser certains objets du script pour les faire référencer des objets de l'application. Chaque langage étant différent, BSF offre une technique générique pour cela.

```
manager.registerBean("myFrame",myFrame);
```

La bridge s'occupant de relier BSF et le langage cible va générer automatiquement un morceau de script pour initialiser une variable `myFrame` référençant l'instance java `myFrame`. Le script peut alors manipuler l'instance java `myFrame` comme s'il s'agissait d'une instance du script.

Par exemple, lors de l'invocation d'un javascript, le code ajouté est le suivant :

```
var myFrame=bsf.lookupBean("myFrame");
...Votre script ici
```

Ce code sera différent pour chaque langage. En TCL, le code sera :

```
set myFrame [bsf lookupBean "myFrame"]
...Votre script ici
```

Quelques informations techniques peuvent être valorisées dans le manageur avant d'invoquer un script. Par exemple, il est possible d'indiquer le répertoire temporaire, le classloader à utiliser ou le `ClassPath` à prendre en compte.

Vous pouvez rédiger votre instance `BSFEngineImpl` pour ajouter un nouveau langage dans le framework. Ainsi, toutes les applications BSF pourront l'utiliser.

Rédigeons un petit bout de code pour invoquer un script dans une servlet.

```
BSFManager mgr=new BSFManager();
mgr.declareBean("application",servlet.getServletContext(),ServletContext.class);
mgr.declareBean("session",request.getSession(false),HttpSession.class);
mgr.declareBean("config",servlet.getServletContext(),ServletContext.class);
mgr.declareBean("request",request,HttpServletRequest.class);
mgr.declareBean("response",response,HttpServletResponse.class);
mgr.declareBean("home",home,String.class);
mgr.exec(language,"test",0,0,script);
```

Voici quelques exemples de scripts inclus dans un fichier XML pour invoquer une page JSP.

En JavaScript :

```
<script language="javascript">
application.getRequestDispatcher(home+"page.jsp").forward(request,response);
</script>
```

En TCL :

```
<script language="jacl">
[$application getRequestMethod [concat [$home toString]page.jsp]] \
    forward $request $response
</script>
```

En Rexx :

```
<script language="netrexx">
application.getRequestDispatcher(home || "page.jsp").forward(request,response)
</script>
```

En Python :

```
<script language="jpython">
application.getRequestDispatcher(home+u"page.jsp").forward(request,response)
</script>
```

En Java :

```
<script language="java">
application.getRequestDispatcher(home+"page.jsp").forward(request,response);
</script>
```

Le bridge java génère un source java dans le répertoire temporaire, puis invoque le compilateur java du JDK. Il charge alors le fichier class en mémoire et l'exécute.

La librairie BSF est maintenant un standard de fait pour l'invocation de script à partir de java. Elle est très facile à utiliser. Vous pouvez ainsi facilement offrir la manipulation de vos applications à partir d'un script, en faisant abstraction du langage utilisé.

Dans certaines situations, il y a quelques difficultés. Par exemple, les langages invoqués par BSF ne savent pas toujours utiliser un classloader différent du classloader système pour invoquer les objets java à partir d'un script. Cela empêche de les utiliser dans un moteur de servlet, car les applications Web n'utilisent pas le classloader système.

Pour le bridge vers le langage java présent avec BSF, le problème vient de la difficulté à initialiser le ClassPath. En effet, ce paramètre est utilisé par le compilateur java pour connaître l'ensemble des classes utilisées par le script. Dans le cadre d'une application Web, le CLASSPATH standard n'est pas correct. Le compilateur n'arrive pas à trouver les classes de l'application.

Certains bridges de langage ne traitent pas toujours correctement l'ensemble des paramètres systèmes comme le classloader ou le répertoire temporaire. J'ai suggéré quelques modifications aux auteurs. En attendant, vous pouvez modifier les sources vous-même !