

JMX

Philippe PRADOS

pp@philippe.prados.name



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

TABLE DES MATIERES

1.	Agent JMX.....	3
2.	Les MBeans	6
2.1	MBean dynamique.....	7
2.2	Open MBean	10

Avant propos

Ce document explique les fonctionnalités de JMX (Java management extension).

Les applications ont besoins d'être administrées à chaud. Il est intéressant de pouvoir intervenir sur le paramétrage de l'application, sans devoir l'interrompre. Par exemple, il doit être possible de modifier à chaud les critères de traces afin d'identifier les problèmes.

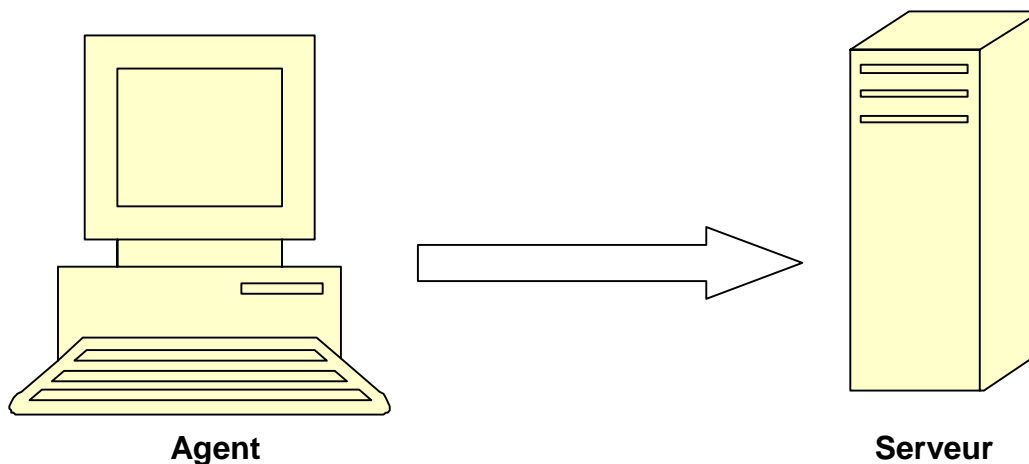
D'autre part, il est intéressant de pouvoir consulter différents indicateurs d'une application afin d'améliorer son comportement. Par exemple, les différents paramètres de caches peuvent être modifiés pour tenir compte d'une pointe de trafic. Des indicateurs applicatifs peuvent avertir d'un risque avant qu'il ne se produit.

Java Management Extension (JMX) propose une technologie normalisée de management. En respectant ces spécifications, une application peut être manager à l'aide d'un agent générique. Les spécifications sont séparées en deux parties. Un pattern de développement pour les applications administrable et un agent s'appuyant sur ce pattern pour manager une application.

L'objectif de JMX est de décharger le développement de l'agent. En quelques lignes, il peut offrir différents indicateurs ou des API permettant de manipuler l'application à chaud. L'agent sera, à terme, intégré aux moteurs J2EE. Cela permettra de manager le moteur lui-même, pour ajouter de nouvelles servlets, modifier les paramètres d'accès aux bases de données, relancer les applications Web ou les moteurs EJB, modifier les paramètres de sécurités, etc.

1. AGENT JMX

En respectant les spécifications JMX, l'agent est capable de manager les autres couches logicielles que vous avez développées.



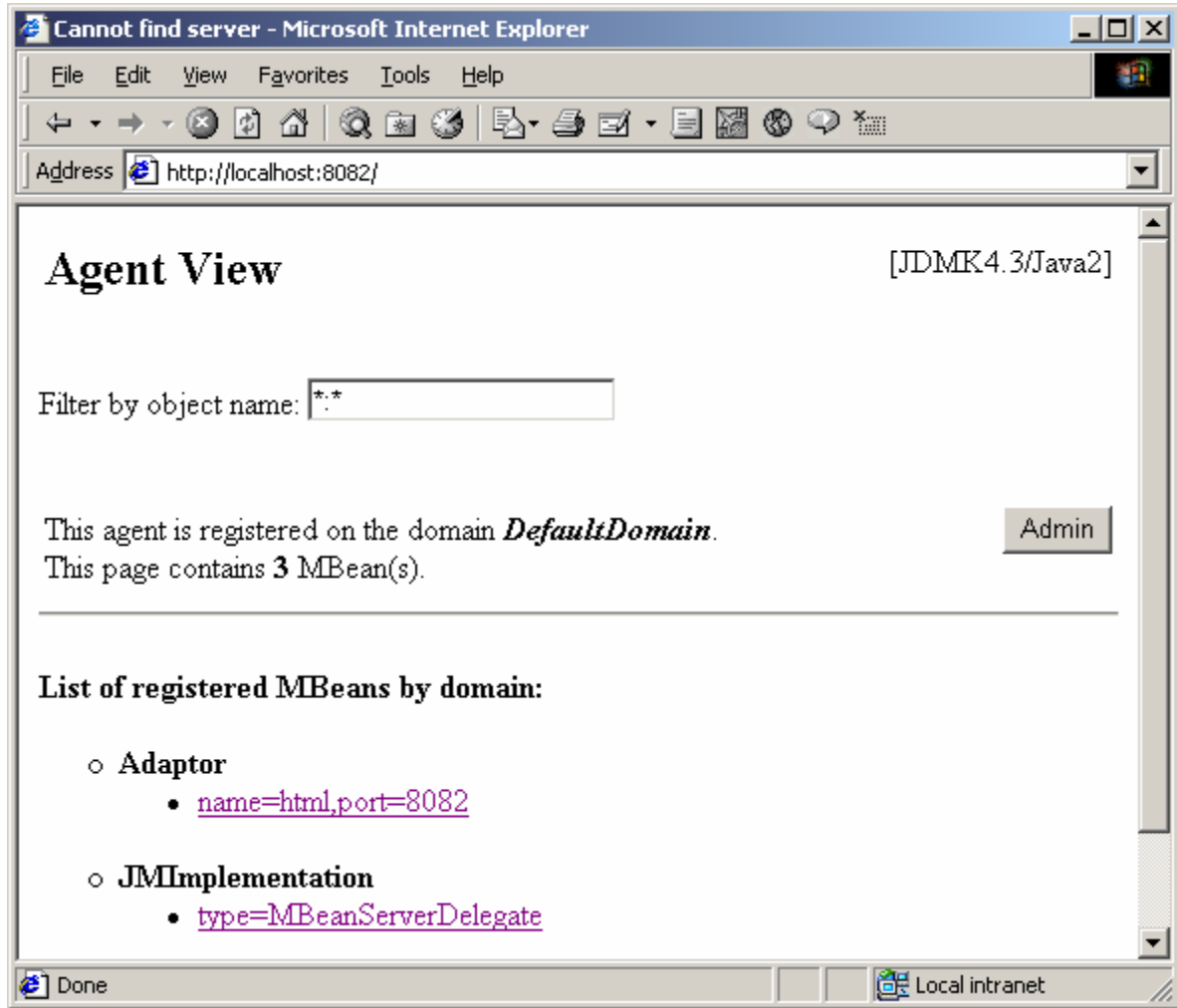
En téléchargeant les spécifications JMX et les bibliothèques associées, un agent est proposé par défaut. Il permet le management d'une application à partir d'un simple navigateur. Cet agent n'offre pas tous les services que l'on peut attendre d'un agent JMX complet. Il n'est pas capable de traiter les événements du serveur afin de rafraîchir la page. Il utilise un mécanisme de pooling pour contourner cela.

Quelques lignes permettent de lancer cet agent.

```
MBeanServer server = MBeanServerFactory.createMBeanServer();
com.sun.jdmk.comm.HtmlAdaptorServer html =
    new com.sun.jdmk.comm.HtmlAdaptorServer();
server.registerMBean(html, new ObjectName("Adaptor:name=html,port=8082"));
html.start();
```

La première ligne crée un serveur de MBean. Un MBean est un bean manageable. La deuxième ligne construit une instance de l'agent HTML. La suivante l'enregistre dans le serveur de MBean sous le nom `Adaptor:name=html,port=8082`. L'agent est lui-même un MBean enregistré. Il est donc administrable. Les noms des MBeans permettent de les retrouver à l'aide de requêtes complexes. La dernière ligne démarre l'agent.

En demandant la page <http://localhost:8082> vous pouvez administrer votre application.



La page affiche le nom des différents MBean enregistrés. Un filtre permet de retrouver un bean particulier à partir de son nom de domaine ou de la valeur d'une de ces propriétés. Le bouton [Admin](#) permet d'enregistrer dynamiquement d'autres MBeans. En cliquant sur un MBean, une page permet de modifier les différentes propriétés ou d'invoquer des traitements.

MBean View [JDK4.3/Java2]

- **MBean Name:** Adaptor:name=html,port=8082
- **MBean Java Class:** com.sun.jdk.com.HtmlAdaptorServer

Reload Period in seconds:

[Back to Agent View](#)

MBean description:

HtmlAdaptorServer class: Provides a management interface of an agent to Web browser clients.

List of MBean attributes:

Name	Type	Access	Value
Active	boolean	RO	true
ActiveClientCount	int	RO	1
AuthenticationOn	boolean	RO	false
Host	java.lang.String	RO	PPRADOS2
LastConnectedClient	java.lang.String	RO	127.0.0.1
MaxActiveClientCount	int	RW	<input type="text" value="10"/>

Si vous désirez protéger l'agent, vous devez indiquer les différents utilisateurs acceptés.

```
MBeanServer server = MBeanServerFactory.createMBeanServer();
AuthInfo[] auth=new AuthInfo[]{new AuthInfo("admin","password")};
HtmlAdaptorServer html = new HtmlAdaptorServer(8082,auth);
server.registerMBean(html, new ObjectName("Adaptor:name=html,port=8082"));
html.start();
```

Le protocole HTTPS n'est pas supporté. Il faut alors faire attention aux différents réseaux traversés lors de l'identification de l'utilisateur. Il est préférable d'administrer l'application directement sur le serveur.

Un autre agent en Open Source résout les limitations de l'agent de Sun. L'agent MX4J permet d'administrer des MBean à partir d'un navigateur HTML, via des filtres XSL. Cela permet d'adapter la présentation à l'application aux MBean manipuler. Il supporte l'utilisation du SSL et améliore la sécurité. Il permet également d'utiliser RMI pour manager les MBean à distance.

MX4J
MX4J/Http Adaptor

Server view MBean View Timers Monitors Relations MLet

MBean DefaultDomain:name=test
Description: Administre MaClass.

Attributes

Name	Description	Type	Value	New Value
state	Etat de MaClass.	int	0	Read-only attribute

Operations

Name	Return Type	Description
reset	void	Reset MaClass.

Constructors

Class	ObjectName	Description	Parameters
MaClass	<input type="text"/>	Administre MaClass	<input type="button" value="create new"/>

Built using [MX4J](#) HttpAdapter

2. LES MBEANS

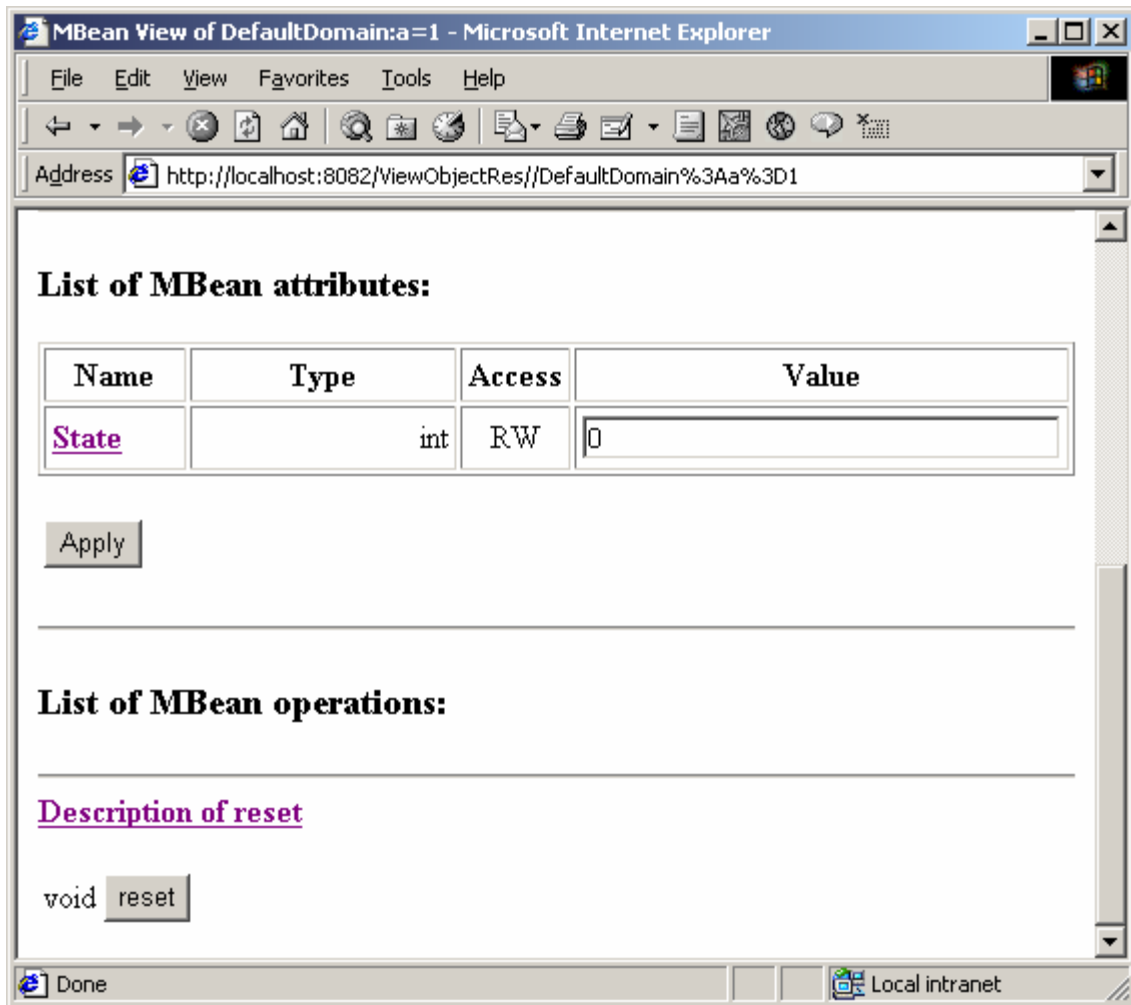
Pour pouvoir être administrable, une application doit enregistrer des MBeans auprès d'un serveur JMX. Un MBean est un bean avec des méthodes, des propriétés et des événements. Il existe différentes catégories de MBean, plus ou moins complexe à rédiger.

Le MBean standard implémente une interface suffixée par MBean, et proposant les différents services et propriété disponibles pour l'agent. Par introspection, le serveur JMX déduit les méthodes et les propriétés du bean.

```
public interface MaClassMBean
{
    public int getState();
    public void setState(int s);
    public void reset();
}
public class MaClass implements MaClassMBean
{
    private int state = 0;
    private String hidden = null;
    public int getState()
    {
        return(state);
    }
    public void setState(int s)
    {
        state = s;
    }
    public void reset()
    {
        state = 0;
    }
}
```

Seul les propriétés et les services décrits dans l'interface sont disponible pour l'agent JMX. En enregistrant une instance `MaClass`, l'agent recherche une interface `MaClassMBean`, l'analyse, et propose la consultation et la modification du bean.

Si vous demandez à l'agent, la création d'une instance `MaClass` sous le nom `:name=test`, vous pouvez modifier la propriété `state` ou invoquer la méthode `reset()`.



En cliquant sur nom de la propriété ou sur la description de la méthode `reset`, vous obtenez un message générique.



Pour offrir des informations plus pertinentes, il faut rédiger un MBean dynamique.

2.1 MBean dynamique

Un MBean dynamique doit décrire son interface à l'aide de méta-donnée et d'une interface spécifique. Cela permet d'enrichir d'un message explicatif la description des propriétés, des méthodes ou des constructeurs. Cela permet également d'informer des différents événements que peut générer le MBean.

L'interface `DynamicMBean` propose différentes méthodes permettant de décrire les possibilités du MBean.

```
public interface DynamicMBean
{
    public MBeanInfo getMBeanInfo();
    public Object getAttribute(String attribute);
    public void setAttribute(Attribute attribute);
    public AttributeList getAttributes(String[] attributes);
    public AttributeList setAttributes(AttributeList attributes);
    public Object invoke(String actionName, Object params[],
        String signature());
}
```

La méthode `getMBeanInfo()` doit retourner toutes les méta-informations décrivant le MBean. Les méthodes `get/setAttribute(s)` permettent de modifier un ou plusieurs attributs. La méthode `invoke` permet d'invoquer une méthode du MBean. Un MBean ne peut implémenter simultanément une interface `nameMBean` et `DynamicMBean`.

Modifions notre MBean pour le rendre dynamique.

```
import javax.management.*;
import java.util.*;

public class MaClass implements DynamicMBean
{
    private int state = 0;
    public int getState()
    {
        return(state);
    }
    public void reset()
    {
        state = 0;
    }
    public void setState(int s)
    {
        state = s;
    }
    //-----
    public MBeanInfo getMBeanInfo()
    {
        MBeanParameterInfo[] noParamInfo=new MBeanParameterInfo[0];
        MBeanAttributeInfo attributes[] = new MBeanAttributeInfo[1];
        attributes[0] = new MBeanAttributeInfo("state","int",
            "Etat de MaClass", true, false,false);

        MBeanConstructorInfo[] constructors = new MBeanConstructorInfo[1];
        constructors[0]=new MBeanConstructorInfo("MaClass",
            "Administre MaClass",noParamInfo);

        MBeanOperationInfo[] operations = new MBeanOperationInfo[1];
        operations[0]=new MBeanOperationInfo("reset",
            "Reset MaClass",
            noParamInfo,
            void.class.getName(),MBeanOperationInfo.ACTION);

        return new MBeanInfo(getClass().getName(),
            "Administre MaClass",
            attributes,
            constructors, // Constructeur
            operations, // Operation
            null); // Notification
    }
    public Object getAttribute(String attribute)
        throws AttributeNotFoundException
    {
        if (attribute.equals("state"))
        {
            return new Integer(getState());
        }
        throw new AttributeNotFoundException(attribute);
    }
    public AttributeList getAttributes(String[] attributes)
        throws AttributeNotFoundException
    {
        AttributeList list=new AttributeList();
        for (int i=0;i<attributes.length;++i)
        {
            String name=attributes[i];
            Object rc=getAttribute(name);
            list.add(new Attribute(name,rc));
        }
        return list;
    }
    public void setAttribute(Attribute attribute)
        throws AttributeNotFoundException, InvalidAttributeValueException
    {
        String name=attribute.getName();
        try
        {
            if (name.equals("state"))
```

```

        {
            setState(((Integer)attribute.getValue()).intValue());
        }
        throw new AttributeNotFoundException(name);
    }
    catch (ClassCastException x)
    {
        throw new InvalidAttributeValueException(name);
    }
}
public AttributeList setAttributes(AttributeList attributes)
    throws AttributeNotFoundException, InvalidAttributeValueException
{
    for (Iterator i=attributes.iterator();i.hasNext();)
    {
        Attribute attr=(Attribute)i.next();
        setAttribute(attr);
    }
    return attributes;
}
public Object invoke(String actionName, Object[] params,
    String[] signature)
    throws MBeanException, ReflectionException
{
    try
    {
        if (actionName.equals("reset"))
        {
            reset();
        }
        return null;
    }
    catch (Exception x)
    {
        throw new MBeanException(x);
    }
}
}
}

```

Maintenant, les descriptions des propriétés et des méthodes sont plus riches. Un clic permet d'avoir la description d'un service ou d'une propriété. Ces services sont suffisants pour l'agent HTML proposé par SUN.

Pour des agents plus complets, nous pouvons générer des notifications lors de la modification de la propriété `state`. Le source suivant indique les modifications.

```

import javax.management.*;
import java.util.*;
import java.beans.*;

public class MaClass extends NotificationBroadcasterSupport
    implements DynamicMBean
{
    private int state = 0;
    private long sequence_;
    ...
    public MBeanInfo getMBeanInfo()
    {
        ...
        return new MBeanInfo(getClass().getName(),
            "Administrateur MaClass.",
            attributes,
            constructors, // Constructeur
            operations, // Operation
            getNotificationInfo()); // Notification
    }
    public final MBeanNotificationInfo[] getNotificationInfo()
    {
        MBeanNotificationInfo[] info;
        info=new MBeanNotificationInfo[1];
        info[0]=new MBeanNotificationInfo(
            new String[]{AttributeChangeNotification.ATTRIBUTE_CHANGE},
            AttributeChangeNotification.class.getName(),
            "Attribut modifié");
        return info;
    }
    public void setAttribute(Attribute attribute)
        throws AttributeNotFoundException, InvalidAttributeValueException
    {

```

```

String name=attribute.getName();
try
{
    if (name.equals("state"))
    {
        int old=getState();
        setState(((Integer)attribute.getValue()).intValue());
        Notification notification=new AttributeChangeNotification(this,
            sequence_++,
            System.currentTimeMillis(),
            "Etat modifié",
            "state",
            Integer.class.getName(),
            new Integer(old),
            attribute.getValue());
        sendNotification(notification);
    }
    throw new AttributeNotFoundException(name);
}
catch (ClassCastException x)
{
    throw new InvalidAttributeValueException(name);
}
}
}

```

Un agent plus ergonomique que l'interface HTML peut afficher des courbes sur l'évolution dans le temps des valeurs des attributs par exemple.

2.2 Open MBean

Un Open MBean est un MBean dynamique n'utilisant que certaines classes pour les attributs et les paramètres des méthodes. Cela permet aux agents de mieux comprendre les informations manipulées par le MBean et de proposer des ergonomies adaptées. Les types acceptés sont les versions classes des types primitifs plus trois nouvelles classes :

- `javax.management.ObjectName`
- `javax.management.openmbean.CompositeData` (interface)
- `javax.management.openmbean.TabularData` (interface)

Celles-ci permettent de décrire des structures complexes ou tabulaire. L'agent peut alors proposer des tableaux ou des arbres pour modifier ou consulter des propriétés.

Notre MBean est déjà un Open MBean car les méthodes de l'interface `DynamicMBean` utilisent le type `Integer`.

En quelques lignes, vous pouvez offrir une administration à chaud de votre application. L'agent par défaut est sympathique, mais n'offre pas toutes les fonctionnalités des MBeans. Les notifications ne sont pas gérées.

JMX est un framework très simple. Des évolutions permettront l'administration à distance de MBean lors de la publication de JMX 1.5. Cette API prendra tout son sens lorsque les moteurs J2EE proposeront un agent JMX complet. Votre application pourra être manager en même temps que vos servlets. Vous pouvez dès maintenant anticiper ces évolutions en enregistrant vos MBean auprès d'un MBean Serveur et en installant un agent HTML.

Spécification : <http://java.sun.com/jmx>

MX for java : <http://mx4j.sourceforge.net/>