Java Management eXtension (JMX)

contenu de la section

Quelques généralités

l'administration

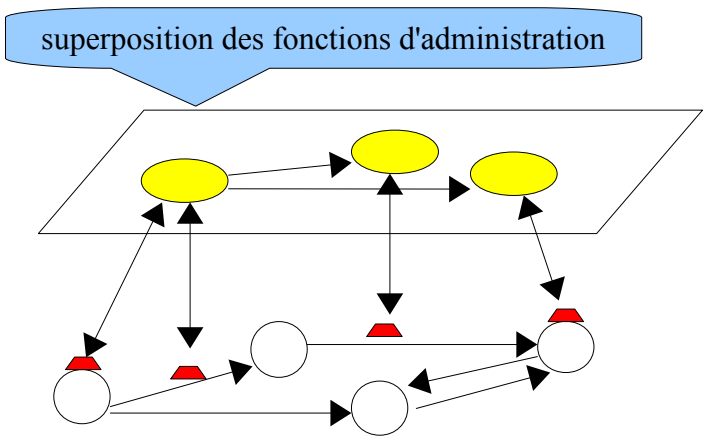
- les actions non directement liées au "métier" de l'application mais agissant sur l'application elle même (garanties de QoS) : la supervision, la configuration (dynamique), l'audit, ...

les objectifs et les principes de JMX

- standardiser une architecture, des design patterns, une API pour :
  - séparer le code d'administration du code "métier"
  - minimiser les interventions sur le code "métier"
  - superposer de façon "transparente" l'administration

- JMX est intégré au jdk1.5

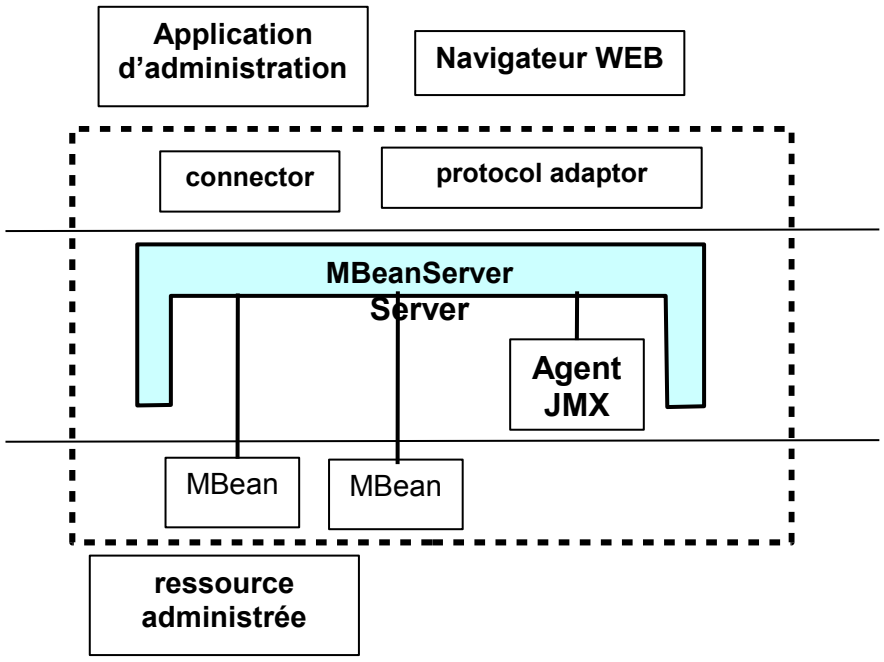
- l'idée directrice : insérer dans la couche "métier"
  - des capteurs fournissant les informations à la couche d'administration
  - des effecteurs répercutant sur la couche "métier" les décisions de la couche d'administration
- l'assemblage des couches "métier" et d'administration repose sur des événements, des appels de méthodes



Le modèle de MBeans

les principaux concepts

- la ressource administrée (administrable)
- l'objet pour interfacer les ressources administrées avec le code d'administration (**MBean**)
- le serveur donnant accès aux **MBeans** (un **MBeanServer**)
- les objets techniques de communication : connector, protocol adaptor (plusieurs spécialisations : SNMP, CIM/WBEM)
- le code d'administration



les MBeans

- un MBean possède :
  - des attributs (lecture-écriture, lecture seule, écriture seule)
  - des opérations
  - des méta-données d'auto-description (MBeanInfo)
- un **MBean** peut émettre des notifications

plusieurs modes de réalisation : standard MBeans, les dynamic MBeans, les open MBeans, les model MBeans

le MBeanServer

- un MBeanServer permet :
  - d'enregistrer/désenregistrer un **MBean**
  - d'accèder aux attributs, aux méthodes aux méta-données d'un **MBean**
  - d'enregistrer/désenregistrer un listener de notifications

Les MBeans (1)

la définition d'un MBean standard

- un MBean est un objet obéissant aux conventions suivantes :
  - il implémente une interface : **<nomBean>MBean**
  - les getters des attributs ont pour forme : **<typeAttribut> get<NomAttribut>()**
  - les setters des attributs ont pour forme : **void set<NomAttribut>(<typeAttribut> arg)**

cas des attributs booléens :
boolean has<NomAttribut>()
boolean is<NomAttribut>()

cas des attributs booléens :
void has<NomAttribut>(boolean arg)
void is<NomAttribut>(boolean arg)

un exemple simple

```
package mbeans;

public interface ExempleMBean {
    public void operation1(String nm);
    public void operation1();
    public String operation2();
    public String getReadOnly();
    public void setReadWrite(String nm);
    public String getReadWrite();
}
```

contrainte sur le nommage
de la classe d'implémentation

```
package mbeans;

public class Exemple implements ExempleMBean {
    private String ReadOnly="Archibald";
    private String ReadWrite;

    public void operation1(String a0) { System.out.println("hello " + a0); }
    public void operation1() { operation1(ReadOnly); }
    public String operation2() { return ReadWrite + ReadOnly; }

    public String getReadOnly() { return  ReadOnly; }
    public synchronized void setReadWrite(String rw) {  ReadWrite = rw; }
    public String getReadWrite() { return ReadWrite; }
}
```

Les MBeans (2)

le nommage des MBeans

- le nom identifie un MBean de façon unique dans un MBean Server
- il comporte un nom de domaine (optionnel) suivi d'une liste de (property=value)

[nom de domaine]:property=value[,property=value]*

- quelques exemples : **//example.mBeans:type=std,user=fho,complexity=0**
  monitoring.component.application:role=supervision

- la classe correspondante est :

```
public class ObjectName {
   public ObjectName(String nm);
   public String getDomain();
   public Hashtable getKeyPropertyList();
   public String getKeyProperty(String nm);
   .........
}
```

Le serveur de MBeans (1)

l'obtention d'un MBeanServer

un MBeanServer permet :
- d'enregistrer/désenregistrer un MBean
- d'accèder aux attributs, aux méthodes et aux méta-données d'un MBean
- d'enregistrer/désenregistrer un listener de notifications

- elle est réalisée via une classe "fabrique" : **MBeanServerFactory**

```
public static MBeanServer createMBeanServer();
public static MBeanServer createMBeanServer(String name);

public static MBeanServer findMBeanServer(String name);

public static MBeanServer newMBeanServer();
public static MBeanServer newMBeanServer(String name);
```

- depuis jdk1.5, un MBeanServer est implicitement associé à toute JVM démarrée
  il est accessible via la méthode static **getPlatformMBeanServer** de la classe **ManagementFactory**

```
public static MBeanServer getPlatformMBeanServer();
```

méthode à utiliser pour enregistrer des MBeans accessibles via la jconsole

l'interface MBeanServer

- elle offre des méthodes d'enregistrement, de désenregistrement, d'un MBean enregistré

```
ObjectInstance registerMBean(Object arg0, ObjectName arg1) throws ....;
void unregisterMBean(ObjectName arg0) throws ......;
```

- elle offre des méthodes d'accès aux attributs, aux opérations, ….. d'un MBean

```
Object getAttribute(ObjectName arg0, String arg1) throws ......;
AttributeList getAttributes(ObjectName arg0, String[] arg1) throws ......;
void setAttribute(ObjectName arg0, Attribute arg1) throws .......;
void setAttributes(ObjectName arg0, AttributeList arg1) throws ........;
public Object invoke(ObjectName arg0, String opName, Object[] parms, String[] signs) throws .......;
```

Le serveur de MBeans (2)

un exemple simple …

```
package mbeans;

public interface ExempleMBean {
    public void operation1(String nm);
    public void operation1();
    public String operation2();
    public String getReadOnly();
    public void setReadWrite(String nm);
    public String getReadWrite();
}
```

```
package mbeans;

public class Exemple implements ExempleMBean {
    private String ReadOnly="Archibald";
    private String ReadWrite;

    public void operation1(String a0) { System.out.println("hello, " + a0); }
    public void operation1() { operation1(ReadOnly); }
    public String operation2() { return ReadWrite + ReadOnly; }

    public String getReadOnly() { return  ReadOnly; }
    public synchronized void setReadWrite(String rw) {  ReadWrite = rw; }
    public String getReadWrite() { return ReadWrite; }
}
```

```
package mbeans;

import java.lang.management.*;
import javax.management.*;
    ..................
    name = new ObjectName("jap.mbeans:type=exemple1");
    Exemple mbean = new Exemple();
    mbs.registerMBean(mbean, name);
    System.out.println("le champ rw de Exemple est " + mbs.getAttribute(name,"ReadWrite"));
    mbs.setAttribute(name,new Attribute("ReadWrite", "Zorgh"));
    mbs.invoke(name,"operation1",new Object[] { "mr Dupont" }, new String[ ] { "java.lang.String" });
    ..........
```

L'accès aux MBeans

la jconsole

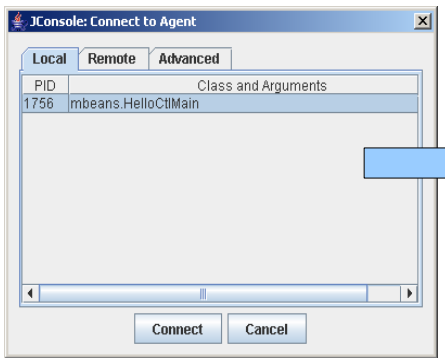
- elle offre un accès au MBeanServer associé à la jvm et aux mbeans enregistrés
- un prototype dont la pérennité n'est pas garanti
- l'interface a évolué en java 1.6

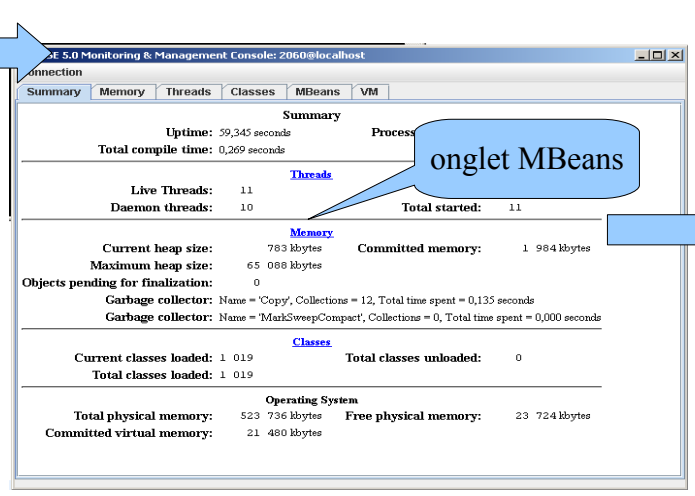
```
public class ExempleMain {
   public static void main(String[] args) {
      Exemple mbean = new Exemple();

      MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
      ObjectName name = new ObjectName("jap.mbeans:type=exemple1");
      mbs.registerMBean(mbean, name);
      try { Thread.sleep(10000000); } catch(InterruptedException e) {}
   }
}
```

```
>java -Dcom.sun.management.jmxremote mbeans.ExempleMain
> jconsole
```

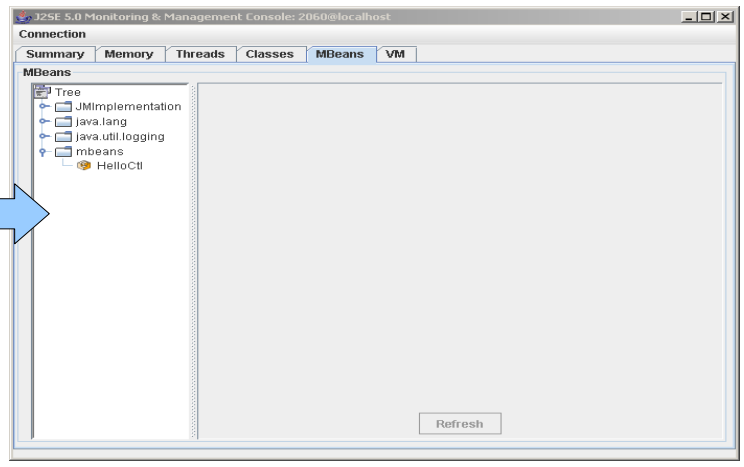
```
>java -Dcom.sun.management.jmxremote mbeans.Exemplemain
> jconsole 1578
```

pid obtenu par jsp ou Task Manager





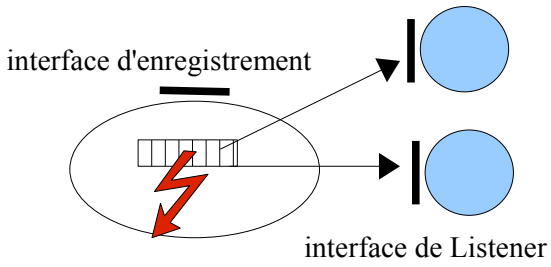
onglet MBeans



Les notifications (1)

le modèle

- un modèle événementiel classique : Notification et NotificationListener
- on peut enregistrer via le MBeanServer, des listeners dans un MBean qui implémente NotificationBroadCaster



les notifications

- Une notification (classe **Notification**) contient les attributs suivants : source, timestamp, message, user data, type, sequence number, ...
  Object getSource(), void setSource(Object obj), long getTimestamp(), void setTimestamp(long ts);
  String getMessage(), void setMessage(), Object getUserData(), void setUserData(Object obj);
  String getNotificationType(), void setNotificationType(String nt), ...;
- quelques sous-classes de **Notification : AttributeChangeNotification, MBeanServerNotification, JMXConnectionNotification**

l'interface NotificationBroadcaster et la classe NotificationBroadCasterSupport

- l'interface offre des méthodes d'enregistrement/désenregistrement
  void addNotificationListener(NotificationListener listen, NotificationFilter filter, Object context);
  void removeNotificationListener(NotificationListener listen);
- la classe implémente l'envoi d'une notification aux listeners enregistrés
  void sendNotification((Notification notif);

Les notifications (2)

l'interface MBeanServer (suite)

- elle offre des méthodes d'ajout/retrait de listeners à un Mbean enregistré

```
void addNotificationListener(ObjectName arg0, NotificationListener arg1, NotificationFilter arg2, Object arg3) throws .....;
void removeNotificationListener(ObjectName arg0, NotificationListener arg1) throws .......;
void removeNotificationListener(ObjectName arg0, NotificationListener arg1, NotificationFilter arg2, Object arg3) throws ....;
.........
```

les interfaces NotificationListener et NotificationFilter

- la notification n'est envoyée que si le retour de isNotificationEnabled() vaut true

```
public class Notificationlistener {
    void handleNotification(Notification notif, Object context);
}
```

- la notification n'est envoyée que si le retour de isNotificationEnabled() vaut true

```
public interface NotificationFilter {
    boolean isNotificationEnabled(Notification not) ;
}
```

Les notifications (3)

un exemple simple

- supervision de l'affichage de la clase Hello par envoi d'une notification

```
package mbeans;

public interface ExempleMBean {
    public void operation1(String nm);
    public void operation1();
    public String operation2();
    public String getReadOnly();
    public void setReadWrite(String nm);
    public String getReadWrite();
}
```

```
public class ExempleListener implements NotificationListener {

    public void handleNotification(Notification arg0, Object arg1) {
        System.out.println("invocation de operation2 :  " + arg0.getTimeStamp());
    }
}
```

```
package mbeans;

public class Exemple extends NotificationBroadCasterSupport implements ExempleMBean  {
    private String ReadOnly="Archibald";
    private String ReadWrite;

    public void operation1(String a0) { System.out.println("hello, " + a0); }
    public void operation1() { operation1(ReadOnly); }
    public String operation2() {
        sendNotification(new Notification("sayHello", this, 0));
        return ReadWrite + ReadOnly;
    }
    public String getReadOnly() { return  ReadOnly; }
    public synchronized void setReadWrite(String rw) {  ReadWrite = rw; }
    public String getReadWrite() { return ReadWrite; }
}
```

la notification est envoyée à tous les listeners enregistrés

Les notifications (4)

un exemple simple (suite)

```
package mbeans;

public class ExempleMain2 {
  public static void main(String[] args) {
     Exemple mbean = new Exemple();

     MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
     ObjectName name = new ObjectName("jap.mbeans:type=exemple2");
     mbs.registerMBean(mbean, name);
     mbs.addNotificationListener(name, new ExempleListener(), null, null);
     try { Thread.sleep(1000 * 600); } catch(InterruptedException e) {}
   }
}
```

```
>java -Dcom.sun.management.jmxremote mbeans.ExempleMain2
> jconsole
```

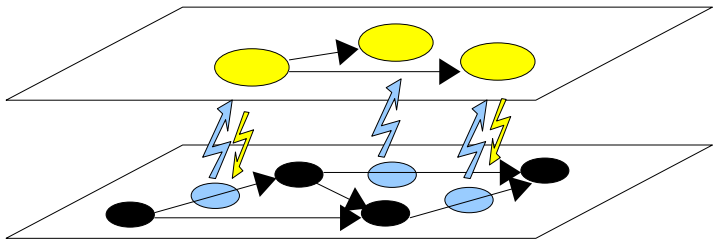
```
>java -Dcom.sun.management.jmxremote mbeans.ExempleMain2
> jconsole 1578
```

pid obtenu par jps ou Task Manager

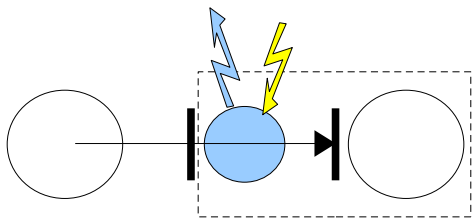
L'administration par les MBeans (1)

une architecture possible

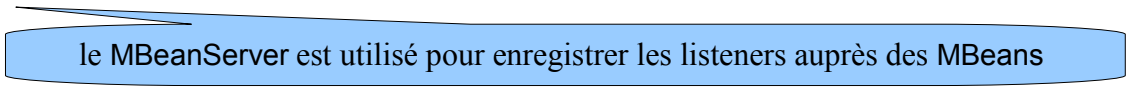
- la superposition de l'administration à l'application repose sur un modèle événementiel
- les MBeans implémentent les senseurs (et les actionneurs) qui génèrent des événements vers les listeners qui implémentent le code d'administration



- les objets "métier" sont les ressources administrables
  - observation et contrôle sur les attributs mais PAS sur les interactions



- les MBean sont des intercepteurs sans sémantique qui :
  - délèguent les appels "fonctionnels"
  - implémentent dans la mesure du possible un mécanisme générique d'observation et de contrôle des interactions (par le blocage/déblocage des délégations d'appel) !!!
  - communiquent avec le code d'administration par événements (Notification)



- le code d'administration

- il est souhaitable de :
  - masquer l'introduction des MBean intercepteurs (usage d'interface, inversion de contrôle, factory-method)
    - distinguer les interfaces "fonctionnelles" (métier) et les interfaces d'administration

L'administration par les MBeans (2)

un exemple simple

- l'application : un client utilisant un objet de la classe **Afficheur** ;

```
public class Client extends Thread {
   private IAfficheur affich;
   private String[ ] texts;

   public Client(IAfficheur af, String[] txt) {
      affich = af;
      texts = txt;
   }

   public void run() {
        for (int i = 0; i < 5000; i++) {
          try { Thread.sleep(2000);  catch(Exception e) { }
           int v = (int) (1000 * Math.random());
           String tx = texts[v % texts.length];
           affich.affiche(tx);
        }
   }
}

public class LanceurAfficheur {
   public static void main(String[ ]args) {
        Client clt = new Client(new Afficheur(), args).start();
   }
}
```

```
public interface IAfficheur {
   public void affiche(String txt);
}

 public class Afficheur implements IAfficheur {
   private String prefix= "affich : ";

   public void affiche(String txt) {
      System.out.println(prefix +txt);
   }
 }
```

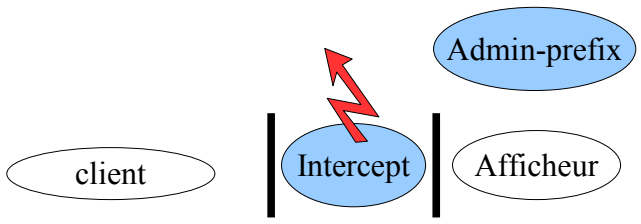
L'administration par les MBeans (3)

l'administration

- observer-paramétrer le préfixe
- faire des statistiques sur le délai séparant 2 affichages successifs
- faire des statistiques sur les chaînes affichées
- introduire un mécanisme de suspend-resume sur l'afficheur ....

- l'administrabilité des ressources
  - le niveau d'administrabilité offert par l'interface n'inclut pas l'observation et du préfixe
  - il est possible d'accéder au champ prefix via la réflexion (si la politique sécuritaire l'autorise)

l'architecture



- l'administration est réalisée via 2 MBeans "capteurs-effecteurs" :
  - un intercepteur pour l'administration qui relève des interactions entre le client et l'afficheur
    un mécanisme d'observation (génération de notification) + le mécanisme de suspend-resume
  - un objet pour l'administration de prefix

  - les capteurs-effecteurs sont sans sémantique applicative ni admministrative
  - les tâches d'administration-supervision sont réalisées dans des listeners (éventuellement réalisés par des MBeans)

L'administration par les MBeans (4)

l'architecture (suite)

- interface pour l'administration des états internes à l'afficheur
- interface pour l'administration des interactions avec l'afficheur

```
public interface AffichPrefixMBean {
    public void setPrefix(String);
    public String getPrefix();
}
```

```
public interface AffichInterMBean {
    public void suspend();
    public resume();
}
```

- le mécanisme d'observation simple est réalisé via des notifications (les informations spécifiques sont stockées dans un objet attaché aux notifications : MethodInfo

```
public class MethodInfo implements Serializable {
    private String methodName;
    private Object[] parameters;
    private Object result;

    public MethodInfo(String mNm, Object[ ] args, Object res ) {
        methodName = mNm; parameters = args; result = res;
    }

    public MethodInfo(String mNm, Object[ ] args ) { this(mNm, args, null); }
    public String getMethodName() { return methodName; }
    public Object[] getParameters() { return parameters; }
    public Object getResult() { return result; }
}
```

- Si on souhaite pouvoir observer les notifications comportant **MethodInfo**, il faut :
  - **MethodInfo implements Serializable (la jconsole comunique via rmi)**
  - lancer la jconsole avec :
    jconsole -J-Djava.class.path="%JAVA_HOME%\lib\jconsole.jar;%JAVA_HOME%\tools.jar;<USER_CLASSPATH>

classpath permettant d'accéder à MethodInfo

L'administration par les MBeans (5)

un exemple simple (suite)

```
public class AffichInter extends NotificationBroadCasterSupport implements IAfficheur, AffichInterMBean, Serializable
{
   private boolean suspended;
   private IAfficheur target;
   private int seqN;

   public AffichInter(IAfficheur tgt) { target = tgt; }

   public void affiche(string txt) {
     if (suspended) synchronized (this) {
        try {  wait(); } catch(InterruptedException e) {}
     }
     seqN++;
     Notification notif = new Notification("invocation", this, seqN);
     Object info = new MethodInfo("affiche",new Object[] { txt });
     notif.setUserData(info);
     sendNotification(notif);
     target.affiche(txt);
   }

   public synchronized void suspend() { suspended = true; }

   public synchronized resume() {  suspended = false;  notifyAll(); }
}
```

```
public class AffichPrefix implements AffichPrefixMBean
{
  private Afficheur target;
  private Field field;

  public AffichPrefix(Afficheur tgt) {
    target = tgt;
    try {
       field = tgt.getClass().getDeclaredField("prefix");
    } catch(Exception e) {}
  }

  public String getPrefix() {
    field.setAccessible(true);
    String res = (String) field .get(target);
    field.setAccessible(false) ;
    return res;
  }

  public void setPrefix(String pf() {
    field.setAccessible(true);
    field.set(target, pf);
    field.setAccessible(false);
  }
}
```

L'administration par les MBeans (6)

un exemple simple (suite)

- le code d'administration

```
public interface InvocationStatisticsMBean {
    public double getInterInvocationMeanTime();
    public double getInterInvocationStdDeviation();
}
```

```
public class InvocationStatistics implements NotificationListener,  InvocationStatisticsMBean {
    private long lastDate;
    private double somme, somme2;
    private int nbObservations;

    public void handleNotification(Notification notif,  Object arg1) {
      if (!notif.getType().equals("invocation") return;
      if (++nbObservations == 1) { lastDate = arg0.getTimeStamp()); return; }
      long olD = lastDate;
      lastDate = arg0.getTimeStamp());
      somme + = lastDate -olD;
      somme2 +=  (lastDate -olD) * (lastDate -olD);
    }

    public double getInterInvocationMeanTime() { return somme/nbObservations; }
    public double getInterInvocationStdDeviation() {
      double mt = getInterInvocationMeanTime();
      return somme2/nbObservations – mt * mt;
    }
}
```

L'administration par les MBeans (7)

un exemple simple (suite)

- le code d'administration (suite)

```
public interface TextStatisticsMBean {
    public String[] getTexts();
    public int getTextStatistics(String txt);
}
```

```
public class TextStatistics implements NotificationListener,  TextStatisticsMBean {
   private Hashtable statistics = new Hashtable();

   public void handleNotification(Notification notif,  Object arg1) {
        if (!notif.getType().equals("invocation") return;
        String txt = ((MethodInfo) notif.getUserData()).getParameters()[0];
        Integer integ = statistics.get(txt);
        if (integ == null) statistics.put(txt, new Integer(1));
        else statistics.put(txt, integ + 1));
   }

   public String[] getTexts() {
        Set keys = statistics.keySet();
        return (String[]) keys.toArray();
   }

   public int  getTextStatistics(String txt) {
        Integer nb = statistics.get(txt);
        if (nb == null) return 0;
        else return nb.intValue();
   }
}
```

L'administration par les MBeans (8)

un exemple simple (suite)

- le lancement de l'application

création de l'administrateur de préfix. Enregistrement dans le MBeanServer

création de l'intercepteur. Enregistrement dans le MBeanServer

création des objets d'administration. Enregistrement dans le MbeanServer. Enregistrement dans l'intercepteur

création et démarrage du client

```
public class AfficheurMain {
   private static String[] texts = { "toto", "titi", "tutu", "tata", "tete" };

   public static void main(String[] args) {
      IAfficheur affich = new Afficheur();

      MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

      AffichPrefix pref = new AffichPrefix(affich);
      ObjectName name3 = new ObjectName(afficheur:type=Afficheurprefix");
      mbs.registerMBean(pref, name3);

      IAfficheur interc = new AffichInter(affich);
      ObjectName name = new ObjectName(afficheur:type=AfficheurIntercepteur");
      mbs.registerMBean(interc, name);

      InvocationStatistics adm1 =  new InvocationStatistics();
     ObjectName name1 = new ObjectName(type=afficheur:type=invocation_statistics");
      mbs.registerMBean(adm1, name1);
      mbs.addNotificationListener(name,adm1, null, null);
     TextStatistics adm2 = new TextStatistics();
      ObjectName name2 = new ObjectName(type=afficheur:type=text_statistics");
      mbs.registerMBean(adm2, name2);
      mbs.addNotificationListener(name, adm2, null, null);

     new Client(interc, texts).start();
    }
}
```

```
>java -Dcom.sun.management.jmxremote mbeans.AfficheurMain
> jconsole -J-Djava.class.path="%JAVA_HOME%\lib\jconsole.jar;%JAVA_HOME%\tools.jar;."
```

Quelques compléments (1)

Les métadonnées

- représentation réflexive simple des Mbean via des classes MBeanInfo, MBeanOperationInfo, MBeanConstructorInfo, MBeanParameterInfo, MBeanAttributeInfo, MBeanNotificationInfo
- l'interface MBeanServer offre des méthodes d'accès aux méta-données d'un MBean
  MBeanInfo getMBeanInfo(ObjectName arg0) throws ....
- la représentation des types et des signature

B	**byte**
C	**char**
D	**double**
F	**float**
I	**int**
J	**long**
S	**short**
V	**void**
Z	**boolean**
L<nom_de_la_classe> ;	**référence de classe**
[<type>	**tableau de <type>**
(<type_arg>)<type_retour>	**signature de méthodes**

la représentation est celle de java

Attention, ne pas oublier le ; terminal

void func(String str,int[] tabI)	(Ljava/lang/String;[I)V
String func(int index, boolean lig,String[ ])	(IZ[Ljava/lang/String;) Ljava/lang/String;

la signature des constructeurs comporte le V de void

Quelques compléments (2)

les MBeans dynamiques

- Les MBeans dynamiques sont des MBeans "peu typées" :
  ses attributs et opérations n'apparaissent pas dans l'interface

- elles implément l'interface **DynamicMBean** :

```
Object getAttribute(String atName) throws AttributeNotFoundException, ....
AttributeList getAttributes(String[ ] atNames) ) throws AttributeNotFoundException, ....
void setAttribute(Attribute attr) throws AttributeNotFoundException, InvalidAttributeValueException, .....
void setAttributes(Attributelist attrs) throws AttributeNotFoundException, .....
Object invoke(String actName, Object[ ] params, String[ ] sign) throws OperationNotFoundException, .....

MBeanInfo getMBeanInfo()
```

- L'implémentation des méthodes est sous la responsabilité des développeurs

```
package mbeans;

public class HelloCtl implements DynamicMBean {
    public HelloCtl() {}
    ..........
    public Object getAttribute(String atName) throws AttributeNotFoundException, .... { return null; }

    public void invoke(String actName, Object[ ] params, String[ ] sig) throws ..... {
         if ("start".equals(actName) target.start(); else target.stop();
    }
}
```

Quelques compléments (3)

les connecteurs

- un connecteur permet l'accès à des MBeanServers (donc des MBeans distants)
- obtention d'un connecteur via une factory

voir la documentation de javax.management.remote.rmi

```
.....
MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
JMXServiceURL url = new  JMXServiceURL("service:jmx:rmi//jndi/rmi://localhost:9999/server");
JMXConnectorServer cServer = JMXConnectorServerFactory.newJMXConnectorServer(url, null,mBeanServer);
cServer.start();
.....
cServer.stop();
.....
```

les Open MBean et les Model MBean

- Open MBean : Extension des DynamicMBean qui augmente leur capacité réflexive
  - Se traduit par des contraintes additionnelles :
    tous les attributs, arguments d'opération, valeur de retour ont un type défini comme sous-classe de OpenType
    extension des MBeanInfo
- Model MBean : Extension des DynamicMBean
  - implémente des interfaces additionnelles : **PersistentMBean, ..**