JmxPreview

AchimHuegen, Dezember 3 2004

This page describes a HiveMind extension for support of the Java Management Extension (JMX).

Prototype

I have a server up and running where you can get an idea of what the jmx code can do. There are two ways of accessing the server.

1. Point your browser to : http://80.66.6.180:8090

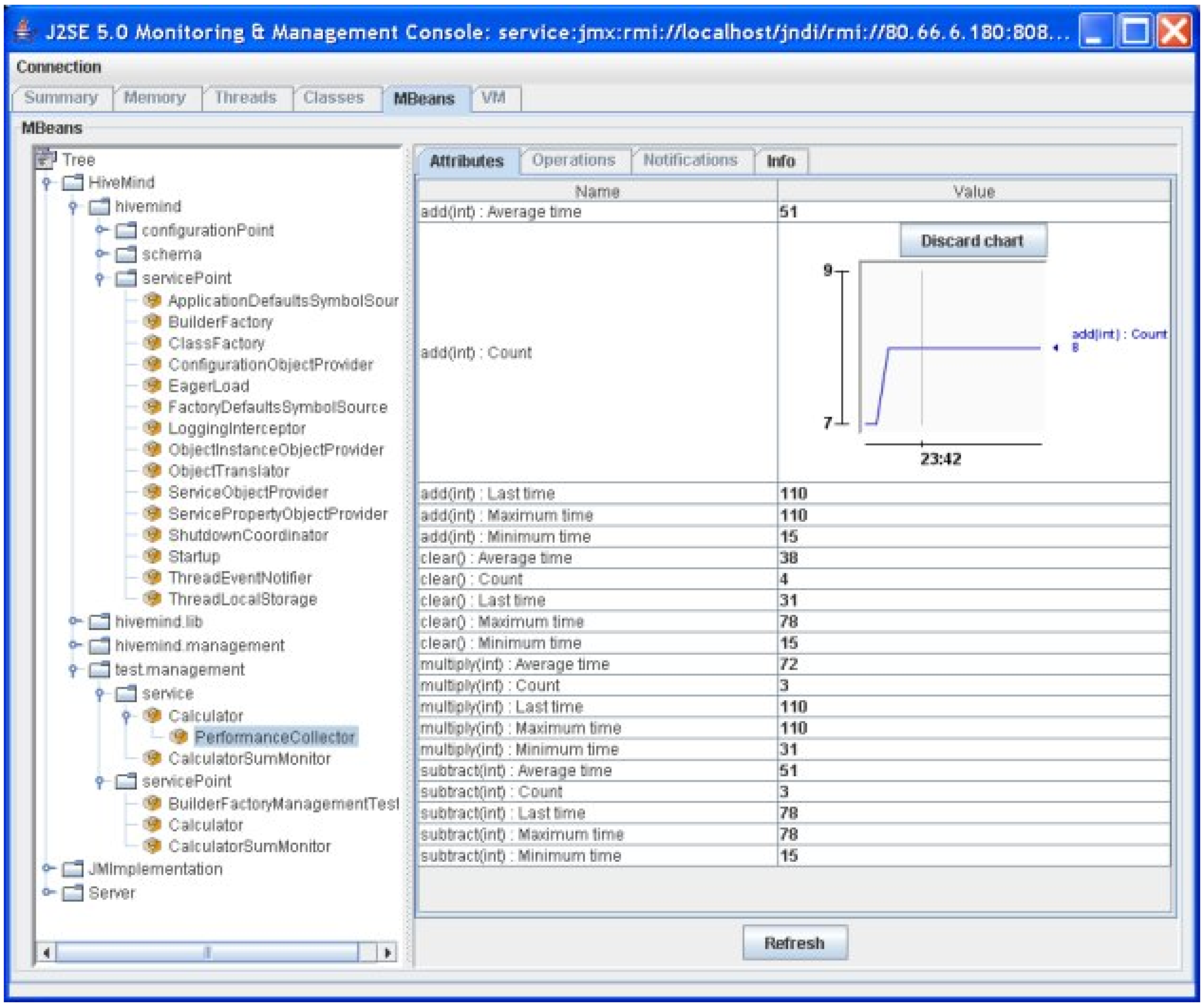
This uses MX4J HttpAdaptor for rendering the management info, but I wouldn't recommend this because MX4J doesn't support tabular data, graphs or a hierarchical display of the mbeans.

2. My favorite: Get Sun's JDK 1.5 and start jconsole. Change to the advanced tab and enter service:jmx:rmi://localhost/jndi/rmi://80.66.6.180:8089/jmx as URL.

JConsole displays the mbeans in a package like tree structure, supports graphs for numerical attributes but has some shortcomings concerning hyperlinks and multiline text attributes.

What you get

The following information assumes, that you are using JConsole. Screenshot of jconsole:



- Tree
  - HiveMind
    - hivemind
      - configurationPoint
      - schema
      - servicePoint
        - ApplicationDefaultsSymbolSour
        - BuilderFactory
        - ClassFactory
        - ConfigurationObjectProvider
        - EagerLoad
        - FactoryDefaultsSymbolSource
        - LoggingInterceptor
        - ObjectInstanceObjectProvider
        - ObjectTranslator
        - ServiceObjectProvider
        - ServicePropertyObjectProvider
        - ShutdownCoordinator
        - Startup
        - ThreadEventNotifier
        - ThreadLocalStorage
    - hivemind.lib
    - hivemind.management
    - test.management
      - service
        - Calculator
          - PerformanceCollector
        - CalculatorSumMonitor
      - servicePoint
        - BuilderFactoryManagementTest
        - Calculator
        - CalculatorSumMonitor
  - JMImplementation
  - Server

Name	Value
add(int) : Average time	51
add(int) : Count	
add(int) : Last time	110
add(int) : Maximum time	110
add(int) : Minimum time	15
clear() : Average time	38
clear() : Count	4
clear() : Last time	31
clear() : Maximum time	78
clear() : Minimum time	15
multiply(int) : Average time	72
multiply(int) : Count	3
multiply(int) : Last time	110
multiply(int) : Maximum time	110
multiply(int) : Minimum time	31
subtract(int) : Average time	51
subtract(int) : Count	3
subtract(int) : Last time	78
subtract(int) : Maximum time	78
subtract(int) : Minimum time	15

Metadata

You can explore all service points, configuration points and schemas defined in the hivemind descriptors. Look for the mbeans of type 'servicePoint', 'configurationPoint' and 'schema'.

This is similar to the information that is created by hivedoc with some bonus:
For example a service point mbean of a service that uses BuilderFactory for creation contains the following data:

- Service implementation class
- Visibility
- Names and values of all properties set during construction. The list contains autowired properties and marks them as autowired.
- The Constructor used
- parameter values of constructor call

You can find the tabular data (e.g. properties values) by double clicking the bold attribute value. A mbean that uses both constructor parameters and properties can be found as test.management.servicePoint.BuilderFactoryManagementTest

The metadata functionality is 'experimental'.

Export services as MBeans

Services can be registered as MBean very easily by a contribution to "mbeans":

```
<contribution configuration-id="hivemind.management.mbeans" >
    <mbean service-id="test.management.Calculator" />
</contribution>
```

Now the complete service interface is available via jmx. (There are some datatype constraints and the servicemodel should be primitive or singleton)

Have a look at the Calculator MBean (test.management.service.Calculator). You can do some calculations on the jconsole operations tab. Double click on the 'sum' attribute to get a graph of the sum over time. This is the corresponding service interface :

{{{public interface Calculator
{
public int getSum();

public void add(int value);

public void subtract(int value);

public void multiply(int value);

public void clear();
}
}}}

Performance interceptor

The interceptor PerformanceMonitor collects statistical data about calls to the intercepted service. The results are sent to a newly created mbean. For each service method it displays: number of calls, maximum, minimum, average and last execution duration.

The mbean test.management.service.Calculator.PerformanceCollector displays these values for the Calculator service. So just execute some calculations and watch the statistical data changing. Double clicking the attribute values displays a graph.

Monitor

Though creation of jmx monitors is not automated a monitor can easily defining as service point using the standard builderfactory functionality. Such a monitor can watch the attribute of another mbean (for example the average execution time of a service method) and send notifications, if a treshold is crossed.

The mbean test.management.service.CalculatorSumMonitor monitors the sum attribute of the calculator service (attribute DerivedGauge). Click the subscribe button on the notification tab to get notified when the calculator sum gets above 100.

Implementation details

Dependencies

The implementation uses JMX 1.2 and supports JSR 160 for remote management (via jconsole, mc4j etc.) It is depending on MX4J but is working fine with Sun RI too. The dependency could be broken up, but there is no standardized way for creating an HTMLAdapter.

Remarks

Services as MBeans

It doesn't always make sense to use the service interface as MBean interface like in the calculator example. It would be nice to separate between service and management interface. Problem: As soon as an service model other that primitive is used or the service is intercepted, the management interface is no longer accessible. The reason is, that interceptors and proxies are limited to a single interface. This is already a problem for standard MBeans, that implement MBean lifecycle events. There are two possible solutions:

- Interceptors and proxies populate multiple/all interfaces of a service. This was already discussed on the mailing list
- There must be a way to access the core service instance