Mit JMeasurement Ausführungszeiten messen

Messung in Produktivumgebungen

VON WILFRIED KLAAS

(zu Version 0.8, 28.02.2010)

Anwendungen im Produktivbetrieb beim Kunden lassen sich leider nur sehr schwer mit Profilingwerkzeugen betreiben. Erstens weil bei manchen die Lizenzierung dieses nicht zulässt, zweitens weil ein Profiling im Produktivbetrieb meistens die Performance so stark beeinflusst, das eine echte Zeitaussage nicht mehr möglich ist. Drittens muss für das Profiling das System neu gestartet werden.

Zeitmessung mit Systemmitteln

Normaleweise sieht eine Zeitmessung mehr oder weniger immer aus wie im Listing 1. Wir nehmen an einer Stelle im Quelltext eine Startzeit und am Ende der Messung geben wir das Ergebnis aus. Für einfache Messungen beim Programmieren mag das durchaus ausreichen. Jedoch sollten solche Konstrukte nicht beim Kunden landen. Die Probleme sind vielfältig:

Listing 1

Typische Zeitmessung

```
//(…)
  long startTime = System.currentTimeMillis();
//(…) zu messender Code
  long stopTime = System.currentTimeMillis();
  System.out.println("Messpunkt 1:"
      + Long.toString(stopTime - startTime));
//(…)
```

- Die Zeitmessung findet immer statt.
- Bei einer Consolenanwendung erscheinen diese Ausgaben ständig und können nicht abgeschaltet werden.
- Man hat keine Möglichkeit, das Ergebnis später vernünftig zu analysieren.

Für Zeitmessung in Produktivsystemen gibt es jedoch eine Anzahl an API's, die jedem Anspruch gerecht werden. In diesem Artikel möchte ich mich mit dem Project JMeasurement beschäftigen. Eine Auswahl an anderen Tools gebe ich später in diesem Artikel.

Das Konzept von JMeasurement

Der Monitor

Im Listing 2 sehen wir den Code mit JMeasurement, der im Prinzip das gleiche tut, wie der Code in Listing 1. Zunächst wird ein Monitor instanziert. Jeder Monitor entspricht einem Messzyklus. Mit

Listing 2

einfache Zeitmessung

```
//(…)
  Monitor monitor = DefaultMonitor()
  monitor.start();
//(…) zu messender Code
  monitor.stop();
  System.out.println(monitor.toString());
//(…)
```

`monitor.start()` wird die Messung automatisch gestartet und mit `monitor.stop()` angehalten. Mit `monitor.asString()` wird dann das Ergebnis der Messung ausgegeben.

Ein Monitor hat noch weitere Möglichkeiten, z.B. kann ein Monitor pausieren und wieder weiter messen. (`pause()` und `resume()`). Man kann die gemessene Zeit künstlich verlängern oder verkürzen. Ein Monitor dient also dazu eine Messung zu machen.

Der Messpunkt

Das nächste wichtige Konzept ist der Messpunkt. (`MeasurePoint`). Ein Messpunkt ist eine Quelltextmarke die immer wieder durchlaufen wird. Somit können an einem Messpunkt mehrere Messungen gemacht werden. Ein Messpunkt wird über seine ID identifiziert. Ein Messpunkt verwaltet automatisch alle seine Monitore, d.h. man kann über einen Messpunkt einen Monitor anfordern. Wird der Monitor gestoppt, wird sein Ergebnis zu dem Messpunkt gerechnet. Der Messpunkt bildet automatisch zusätzliche Informationen, z.B. Anzahl der

Aufrufe, Mittelwert, Gesamt-, kürzeste und längste Zeit. Weiterhin kann der Messpunkt feststellen, ob ein Monitor „gestorben“ ist, d.h. der Monitor wurde zwar gestartet, aber nicht wieder gestoppt. Die Zeit dieses Monitors geht nicht in die Berechnung ein. (s. Listing 3)

Listing 3

Zeitmessung mit Messpunkt

```
//(…)
  MeasurePoint point = new
      DefaultMeasurePoint("de.mcs.jmeasurement.test");
  Monitor monitor1 = point.start();
  //(…) zu messender Code
  monitor.stop();
  System.out.println(point.toString());
//(…)
```

Die Factory

Die nächste Klammer um die unterschiedlichen Messpunkte ist die Factory. Über die Factory werden übergreifende Funktionen zur Verfügung gestellt. Zunächst bietet die Factory einfache Funktionen zur Erzeugung von Messpunkten und Monitoren an. (s. Listing 4)

Listing 4

Zeitmessung mit Factory

```
//(…)
  monitor =
      MeasureFactory.start("de.mcs.jmeasurement.test");
  //(…) zu messender Code
  monitor.stop();
  System.out.println(MeasureFactory.asString());
//(…)
```

Hierbei wird automatisch ein Messpunkt mit der angegebenen ID erzeugt und ein Monitor für diesen Messpunkt instanziert.

Über die Factory können alle Messpunkte und Monitor angesprochen werden. Weiterhin kann man mit der Factory das komplette Zeitmesssystem ein und ausschalten (`setEnable(boolean)`) Auch kann über die Factory eine Priorität vergeben werden. Nur Messpunkte, deren Priorität über der der Factory liegen werden in die Messung einbezogen. Somit kann man verschiedene Level von Messpunkten erzeugen. Messpunkte, die nicht verwendet werden sollen, liefern einen `NullMonitor` zurück. Dieser Monitor misst einfach nichts.

Weiterhin kann man mit der Factory die aktuellen Daten persistent speichern (z.B. als XML-Datei). Diese kann dann beim nächsten Start der Anwendung wieder eingeladen werden.

Auch steuert die Factory das Rendersystem und die SnapShots.

Renderer

Um einen Report zu erzeugen , werden Renderer verwendet. Über die Factory kann man einen Report mit einem bestimmten Renderer Erzeugen. Im Standard sind 3 Renderer implementiert. Ein CSV Renderer, der eine CSV Ausgabe erzeugen kann. Ein HTML und ein einfacher Text Renderer. HTML und Textrenderer können auch die SnapShots ausgeben, die beim CSV-Renderer ignoriert werden.

Die Benutzung eines Renderers ist einfach. (s. Listing 5)

Für besondere Fälle kann natürlich auch ein eigener Renderer verwendet werden.

Callbacks

Bei verschiedenen Operationen können Callbacks registriert werden. Z.B. in der Factory kann ein MeasureDataCallback registriert werden. Mit Hilfe dieses Callbacks können die Daten eines Monitors und des zugehörigen Messpunktes noch modifiziert werden, bevor die Daten des Monitors in den Messpunkt übertragen werden. Der Callback ist z.B. auch geeignet, um über einen Loggingmechanismus jeden Monitor zu protokollieren.

Mit Hilfe von IUserData können einem Messpunkt zusätzliche Benutzerdaten hinzugefügt werden. Diese werden dann im Report über den Renderer mit ausgegeben.

Interfaces

Eine Besonderheit von JMeasurement ist die automatische Messung von Interfacemethoden. Dazu wird einfach das Interface (bzw. die Klasse, die das Interface implementiert) in der Factory registriert. Und schon werden alle Methoden Aufrufe des Interfaces gemessen. Die Messpunkte bekommen automatisch einen Namen, vollständiger Klassenname + # + Methodennamen. (s. Listing 6). Zusätzlich kann man angeben, ob Exceptions mit protokolliert werden sollen und, wenn ja, ob der Vollständige StackTrace oder nur die Exception selber protokolliert werden sollen. Der letzte Parameter ist ein Array von Strings, mit dem man die Messung auf bestimmte

Listing 5

Ausgabe der Messdaten mit einem Writer

```
//(Messungen…)
  PrintWriter output = new
                       PrintWriter("e:\\temp\\output.html");
  MeasureFactory.getReport(null, new DefaultHTMLRenderer(),
                           output);
  output.close();
//(…)
```

Listing 6

Registrierung eines Interfaces

```
//(…)
   ITestProxy testProxy = (ITestProxy)
      MeasureFactory.registerInterface(new CTestProxy(),
      true, true, null);
//(…)
```

Methodennamen einschränken kann. Bei `null` werden alle Methoden gemessen. Sollen bei bestimmten Methoden auch noch ein Argument der Methode als Name des Messpunktes Verwendung finden, kann man die folgende Syntax für den Methodennamen in dem Array verwenden:

```
delete.arg0
```

Hier wird zusätzlich der erste Parameter der Methode an den o.g. Pointnamen angehangen. Man kann auch verschiedene Parameter hintereinander verwenden:

```
delete.arg0.arg2
```

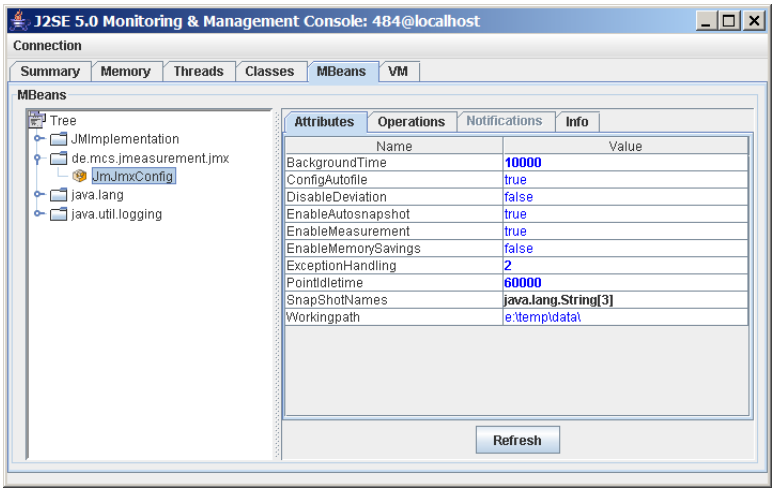
Benutzen kann man die Benamung um z.B. bei SQL Statments die Ausführungszeiten verschiedener Statments auch unterscheidbar zu machen.

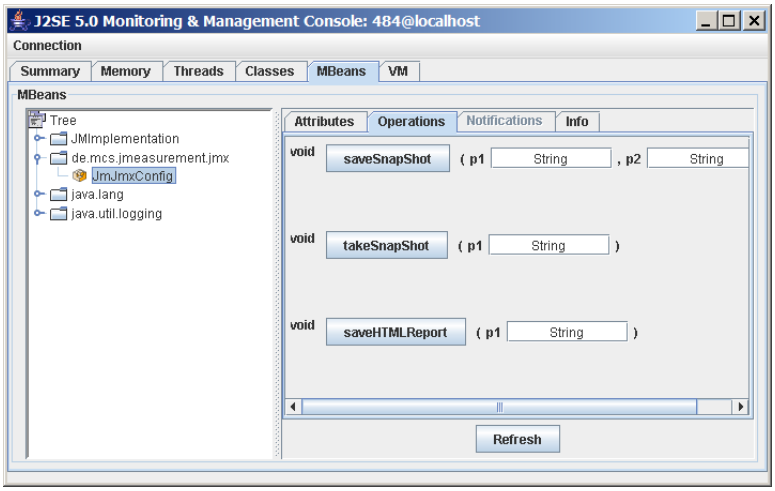
Konfiguration

Die Konfiguration für JMeasurement kann auf verschiedene Weisen durchgeführt werden. Zunächst kann man einfach in der Factory die entsprechenden Methoden benutzen. (z.B. setEnable(), setOption()) Die Konfiguration kann jedoch auch aus einer im Classpath liegenden Datei geladen, sowohl im properties Format wie auch im Properties XML Format. Die Datei muss dann jmconfig.properties bzw. jmconfig.xml heißen. Über configure() wird die Konfigurationsdatei geladen und interpretiert. Ist in der Konfiguration die automatische Überwachung der Konfigurationsdatei parametriert (`OPTION_CONFIG_AUTOFILE`) wird die Datei nun ständig überwacht. Bei Änderungen werden diese dann übernommen. So kann man JMeasurement von „außen“ steuern. Programmatische Änderungen der Optionen haben jedoch Vorrang vor den Einstellungen in der Konfigurationsdatei.

Snapshots

Ein wichtiges Hilfsmittel Zeitverzögerungen auf die Spur zu kommen, ist der SnapShot. Beim Snapshot wird ein komplettes Abbild der aktuell in der Factory vorhandenen Messpunkte eingefroren. Zusätzlich werden ein paar Speicherinformationen mit abgelegt.





Auch die Snapshots können über die Reportfunktionen der Factory mit ausgegeben werden. Der HTML- und der Textrenderer unterstützen Snapshots. Beim CSV Renderer ist das nicht möglich. Stattdessen kann man dort jeden einzelnen Snapshot selber als CVS rendern lassen.

JMX Unterstützung

Seit der Version 0.70 wurde eine JMX Unterstützung integriert. Je nachdem, ob man mit Java 5 oder 6 arbeitet, kann man die Konfiguration von JMeasurement komplett von außen steuern (Java 5) und Messpunkte abfragen (erst ab Java 6). Dazu muss die VM zusätzlich mit folgenden Parametern gestartet werden (Java 5)

```
com.sun.management.jmxremote.port=3456
com.sun.management.jmxremote.authenticate=false
com.sun.management.jmxremote.ssl=false
```

Zusätzlich muss noch im Programm die Factory Methode `registerMBeans` aufgerufen werden. Damit werden die JMX Beans dem Beanserver zur Verfügung gestellt. Nun kann man sich mit einer beliebigen JMX Console auf die VM aufschalten und sieht dann das JMeasurement Config Bean. Unter Java 6 wird ein weiteres Bean angezeigt welches für die Online Abfrage einzelner Messpunkte zuständig ist. Somit kann die Anwendung kontinuierlich überwacht werden.

Spring Unterstützung

Ab Version 0.80 wird auch die Zeitmessung über Spring und AOP unterstützt. Dazu ist einfach nur das entsprechende Advice-Bean einzutragen. Diese erfolgt so:

```
<bean id="advice"
class="de.mcs.jmeasurement.spring.MeasureMethodInterceptor" />
```

Pointcut und Advisor sind wie im Standard zu definieren.

Andere API's

Neben JMeasurement (http://sourceforge.net/projects/jmeasurement2)

gibt es noch eine Vielzahl weiterer API's. Zu erwähnen wäre da das JAMon API (http://jamonapi.sourceforge.net)

JAMon eignet sich mehr für das Messen von Webanwendungen. Es besitzt ein eigenes Servlet für die Ausgaben und kann (auch über den Interfacemechanismus) direkt die Ausführungszeiten von JBDC SQL Anfragen ermitteln.

(Wilfried Klaas, 28.02.2010)