JavaMelody 1.41.0

Eenvoudige monitoring van applicaties en de applicatieserver

Systeemvereisten:

Servlet api 2.4 (of JavaEE 1.4), b.v. Tomcat 5.5, 6 of 7, GlassFish v2 of v3, JBoss 4, 5, 6 of 7, Jonas 4 of 5, Jetty 6 of 7, WebLogic 9, 10 of 11

Ontwikkeld door:	Emeric Vernat (gehost bij Google Project Hosting)
Contactpersoon:	koen.vanderkimpen@smals.be

Open Source (LGPL)

Functionaliteiten

JavaMelody (http://code.google.com/p/javamelody/) is een pakket dat in het algemeen aan Java web applicaties (servlet of Java EE) kan worden toegevoegd om deze en de server waarop ze draaien te monitoren. Specifiek is er extra functionaliteit (d.m.v. configuratie of plugins) bij het monitoren van Hudson, Jenkins, JIRA, Bamboo, Confluence en Liferay servers en voor Grails, Spring, Guice en JSF toepassingen. Bovendien is er een Nagios plugin beschikbaar gesteld door een externe partij (Nagios is een uitgebreider monitoring framework).
De monitoring gebeurt op statistische wijze: de specifieke requests voor de applicatie worden dus niet allemaal opgeslagen. Op deze manier blijft de overhead laag. Daarenboven werkt Javamelody volledig passief: er worden geen extra requests naar de server gestuurd, zoals bij een benchmarking of profiling tool.
Zowel applicatiespecifieke zaken als algemene informatie m.b.t. de applicatieserver worden gemonitored. Zo krijg je b.v. informatie over het aantal, de response tijden en de errors bij http en dieperliggende sql requests, alsook jsp pagina's en business facades, maar ook geheugen- en cpu-gebruik en informatie over threads en garbage collection is beschikbaar. Bovendien kan men ook de volledige lijst van opstart-opties van de Java virtuele machine raadplegen, alsook de (jar) dependencies van de webapplicatie.
Rapportering kan op verschillende manieren: de eenvoudigste is om, in de context van de webapp, naar het adres "/monitoring" te surfen; zo krijgt men de meest recente gegevens (deze pagina moet natuurlijk adequaat worden beveiligd, gezien heel wat van de informatie van het systeem wordt vrijgegeven). Op de html-pagina's kan men de rapporten ook in pdf-formaat downloaden. Daarnaast kan men deze rapporten in pdf naar een bepaald email-adres laten versturen (mits men over een smtp-server beschikt), met een instelbare frequentie (dagelijks tot maandelijks). Alerting (een bericht sturen indien een bepaalde situatie wordt gedetecteerd) hoort echter niet bij de functionaliteiten.
Tot de geavanceerde functionaliteiten van Javamelody behoort een experimentele methode (enkel beschikbaar voor Tomcat) om alle webapplicaties van een server, zonder ze aan te passen, toch te kunnen monitoren. Een andere, reeds beproefde functionaliteit is het opstellen van een zogenaamde "collect server": dit is een centraal systeem dat de monitoring-gegevens van verschillende andere systemen kan opvragen en gecentraliseerd weergeven; de individueel opgeslagen gegevens worden daarbij uit het gemonitorde systeem weggehaald, wat de overhead verder vermindert.

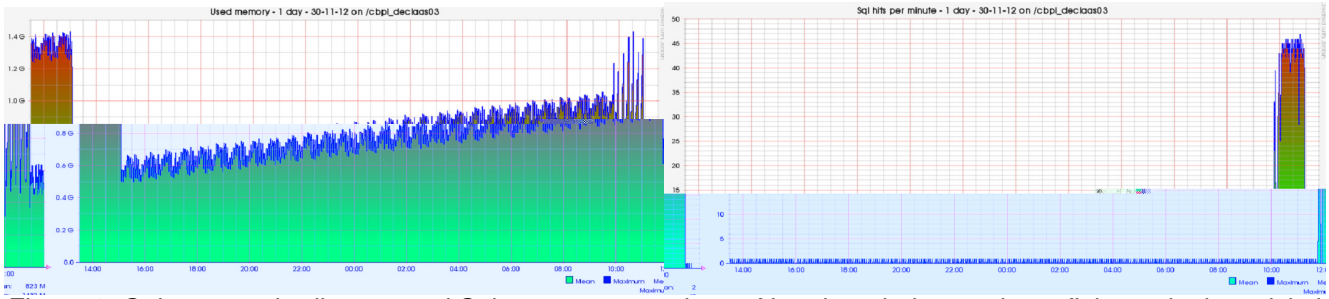
Conclusies en Aanbevelingen

JavaMelody is een handig lichtgewicht product voor het monitoren van webapplicaties en applicatieservers, zowel tijdens ontwikkeling als in productie. Zonder veel overhead laat het nuttige statistieken zien van het systeem, zowel bij normaal als abnormaal gebruik. Het uitmeten van de response tijd van de verschillende types requests die een systeem binnenkrijgt, alsook het relatieve belang van deze requests, kan toelaten om gericht te optimaliseren of om preventief mogelijke problemen aan te pakken.

Wanneer een vollediger oplossing is vereist, inclusief alarmen en het automatisch uitvoeren van stresstests, moet men op zoek gaan naar een uitgebreider pakket, eventueel naast het gebruik van JavaMelody.

Testen en Resultaten

We installeerden JavaMelody in een proof-of-concept applicatie, reeds eerder ontwikkeld bij Onderzoek (de Declaas Companion webapp, draaiende op een Tomcat 6 Server met een MySQL backend). Het pakket is eenvoudig te importeren via een maven-dependency, met een tweede, optionele dependency voor het genereren van de pdf-rapporten. Voor het versturen van emails moesten ook javax.mail en javax.activation in het classpath zitten (typisch staan deze in de libraries van de server). Naast het instellen van de dependency is er nog wat configuratie nodig in web.xml en context.xml van de te monitoren applicatie. Alles bij elkaar was dit op vijftien minuten geregeld.
Daarna volgde monitoring van het systeem gedurende een aantal dagen, waarbij we dagelijks een pdf-rapport lieten versturen en ook af en toe eens gingen kijken naar de monitoring webpagina. Op een bepaald moment lieten we een loadtest los op het gemonitorde systeem om wat actie in de grafieken te krijgen. In de figuren zijn een aantal uittreksels van één van de pdf-rapporten te zien; wegens het quick review formaat is dit heel beperkt moeten blijven. Voor meer screenshots van het product, kan u de volgende pagina raadplegen: http://code.google.com/p/javamelody/wiki/Screenshots.



Figuur 1: Geheugengebruik en aantal Sql requests per minuut. Naar het einde van de grafiek toe, is de activiteit tijdens de loadtest te zien.

Detailed statistics sql - 1 day

Request	% of cumulative time	Hits	Mean time (ms)	Max time (ms)	Standard deviation	% of system error
SELECT * FROM orbeon_form_definition WHERE app=? AND form=? ORDER BY last_modified DESC LIMIT 1;	92	292	19	398	29	0.00
SELECT COUNT(1) AS "count" FROM (SELECT document_id, MAX(last_modified) AS last_modified FROM orbeon_form_data WHERE app=? AND form=? GROUP BY document_id) d1 NATURAL JOIN (SELECT * FROM orbeon_form_data WHERE app=? AND form=?) d2 WHERE deleted='N';	5	42	7	49	9	0.00
SELECT d1.* FROM orbeon_form_data AS d1 NATURAL JOIN (SELECT document_id, MAX(last_modified) AS last_modified FROM orbeon_form_data WHERE app=? AND form=? GROUP BY document_id) d2 WHERE deleted='N' ORDER BY created DESC LIMIT 0,9223372036854775807;	1	42	2	32	4	0.00
SELECT * FROM orbeon_form_data WHERE app=? AND form=? AND document_id = ? ORDER BY last_modified DESC LIMIT 1;	0	4	1	1	0	0.00

Figuur 2: Opsomming van de verschillende queries die door de applicatie naar de database werden verstuurd, met response tijden en relatief belang.

In de tweede figuur is een vrij positief beeld te zien van de response tijden van de sql queries. Dit was in eerste instantie niet zo: JavaMelody liet ons inzien dat een bepaalde query veel te lang duurde; het resultaat in de figuur is datgene na het herschrijven van de desbetreffende query.
We ondervonden slechts één probleem bij het gebruik van JavaMelody: het aantal gebruikte jdbc-connecties bleef zogezegd onbeperkt stijgen in de tijd. Na manueel nazicht bleek dit niet zo te zijn. Dit ligt dus aan een verkeerde berekening binnen het product. Mogelijk zijn we er echter niet in geslaagd dit correct te configureren: onze database resource was heel erg geparametriseerd. Bij gebruik van jndi defaults zouden de standaard instellingen van JavaMelody correct moeten werken.

Gebruiksvoorwaarden