Generic Statistics

Getting Started with gstatistics

This is the getting started guide for the Generic Statistics (gstatistics) component of the grepo framework. It's not supposed to be a complete reference manual - the goal is to show a basic usage and configuration scenario of grepo's gstatistics component. If you have problems understanding parts of this guide or the framework in general or if you have any suggestions, good ideas or if you have found potential bugs please let us know. So let's get started!

> **Version Information**
>
> This is the getting started guide for the Generic Statistics (gstatistics) component of the grepo framework version 1.5.x.

Download the demo project

The demo project for this guide can be checked out from our SVN repository as follows:

```
$ svn checkout
http://svn.codehaus.org/grepo/tags/demo-grep
o-statistics-1.5.0 demo-grepo-statistics
```

The demo project is a maven project and we highly recommend that you use maven to set up the project. If you don't want to use maven you can also set up the project manually. If you use maven and eclipse you can easily make an eclipse project using the following command in the demo project's root directory:

```
$ mvn eclipse:eclipse
```

You can now import the project in your eclipse workspace.

After you have imported the project you should now be able to run the *MyServiceTest* JUnit test. You can also run the test using maven from command line:

```
$ mvn test
```

Demo application

The teeny-weeny demo application consists of one interface (*MyService*) and a class implementing the interface (*MyServiceImpl*). The method *doSomething1()* makes use of grepo's *MethodStatistics* annotation. Method *doSomething2()* demonstrates how to use grepo's *StatisticsManager* directly.

Note: In the demo project we use the gstatistics component to track statistics of method executions. However its also possible to track statistics for various executions (not just method executions), e.g. tracking statistics of entire business processes etc.

Using the grepo framework

In order to use the gstatistics component you need the following grepo artifacts (jars) in your project's classpath:

- grepo-core-<*VERSION*>.jar
- grepo-statistics-<*VERSION*>.jar

Somewhere in your Spring application context (xml) you have to import the default configuration of the grepo gstatistics component.

```
<import
resource="classpath:META-INF/grepo/grepo-sta
tistics-default.cfg.xml"/>
```

In the demo project this is done in *src/main/resources/META-INF/spring/application-context.xml*. Note that you may not need to import that file if you decide to setup grepo with special/custom configuration - you could for instance configure the required *"grepo"* beans directly in your application context.

Furthermore we use spring's component-scan feature in order to create a (spring) bean for our *MyServiceImpl* class - this is standard spring configuration. Additionally we use grepo's *SimpleStatisticsCollectionPrinter* which is used to print out summary and details about collected statistics. (Using the *SimpleStatisticsCollectionPrinter* is optional, but we use its handy methods in our *MyServiceTest* class to print out some information about collected statistics):

```
<bean id="statisticsCollectionPrinter"
class="org.codehaus.grepo.statistics.collect
ion.SimpleStatisticsCollectionPrinter">
    <property name="collection"
ref="grepo.statisticsCollection" />
    <property name="type" value="TXT" />
</bean>
```

That's it, you are now ready to collect statistics using the grepo framework!

Interpreting the output

When running test-method from *MyServiceTest* you should get a summary output similar to the following:

```
STATISTICS SUMMARY

1 collection entries (sorted by
IDENTIFIER_ASC):

demo.service.MyService.doSomething1:
invocations=10 minDuration=10 (Tue Oct 26
15:34:42 CEST 2010) maxDuration=476 (Tue Oct
26 15:34:43 CEST 2010) avgDuration=275
```

This output is produced by grepo's *SimpleStatisticsCollection* printer using the *printSummary()* method. We can see that we have 1 collection entry, which is displayed in the line below. We read the line as follows: The identifier for the collection entry is '*demo.service.MyService.doSomething1*' and there were 10 invocations for that identifier. The minimum duration was 10 milliseconds, the maximum duration was 476 milliseconds, and the average duration is 275 milliseconds.

You should also see detail output similar to the following:

```
STATISTICS DETAIL
```

```
identifier:
demo.service.MyService.doSomething1
invocations: 10
minDuration: 10 (Tue Oct 26 15:34:42 CEST
2010)
maxDuration: 476 (Tue Oct 26 15:34:43 CEST
2010)
avgDuration: 275

5 top durations (sorted by
DURATION_MILLIS_DESC):

 duration: 476 creation: Tue Oct 26 15:34:43
CEST 2010 completion: Tue Oct 26 15:34:43
CEST 2010 origin:
 duration: 431 creation: Tue Oct 26 15:34:43
CEST 2010 completion: Tue Oct 26 15:34:44
CEST 2010 origin:
 duration: 401 creation: Tue Oct 26 15:34:44
CEST 2010 completion: Tue Oct 26 15:34:44
CEST 2010 origin:
 duration: 397 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:43
CEST 2010 origin:
 duration: 363 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:42
CEST 2010 origin:


10 recent invocations (sorted by
```

```
CREATION_DESC):

 duration: 401 creation: Tue Oct 26 15:34:44
CEST 2010 completion: Tue Oct 26 15:34:44
CEST 2010 origin:
 duration: 152 creation: Tue Oct 26 15:34:44
CEST 2010 completion: Tue Oct 26 15:34:44
CEST 2010 origin:
 duration: 431 creation: Tue Oct 26 15:34:43
CEST 2010 completion: Tue Oct 26 15:34:44
CEST 2010 origin:
 duration: 476 creation: Tue Oct 26 15:34:43
CEST 2010 completion: Tue Oct 26 15:34:43
CEST 2010 origin:
 duration: 62 creation: Tue Oct 26 15:34:43
CEST 2010 completion: Tue Oct 26 15:34:43
CEST 2010 origin:
 duration: 397 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:43
CEST 2010 origin:
 duration: 10 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:42
CEST 2010 origin:
 duration: 156 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:42
CEST 2010 origin:
 duration: 308 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:42
CEST 2010 origin:
 duration: 363 creation: Tue Oct 26 15:34:42
CEST 2010 completion: Tue Oct 26 15:34:42
```

```
CEST 2010 origin:
```

Here we can see the details about the collection entry with the identifier '*demo.service.MyService.doSomething1*'. This output is produced by grepo's *SimpleStatisticsCollection* printer using the *printDetail()* method. At the top of the output we see the summary for the collection entry (invocations, min-duration, max-duration, average-duration). Then we see five the top durations (including details) that is invocations which had the longest execution duration. And finally we see the 10 recent invocations (including details).

How it works

Identifier generation

The identifier for the collection entry is generated automatically using an instance of *org.codehaus.grepo.statistics.service.StatisticsEntryIdentifierGenerationStrategy*. In the default configuration grepo uses an instance of *org.codehaus.grepo.statistics.service.StatisticsEntryIdentifierGenerationStrategyImpl* - this implementation generates identifiers as follows:

- **method.getDeclaringClass().getName() + "." + method.getName()**

Feel free to provide your own implementation if desired.

You can also use the *identifier* attribute of grepo's *MethodStatistics* annotation. Doing so grepo won't use the *StatisticsEntryIdentifierGenerationStrategy* at all. When using the *StatisticsManager* directly (as demonstrated in *MyServiceImpl.doSomething2()*) then you have to provide an appropriate identifier anyway, so grepo won't use the *StatisticsEntryIdentifierGenerationStrategy* in that case.

Collecting statistics

The *StatisticsManager* uses a *org.codehaus.grepo.statistics.collection.StatisticsCollectionStrategy* for collecting statistics. In the default configuration grepo uses an instance of *org.codehaus.grepo.statistics.collection.InMemoryStatisticsCollectionStrategy* which stores collections entries in memory using an instance of *org.codehaus.grepo.statistics.collection.StatisticsCollection*. Feel free to provide your own implementation(s) if desired.

The *org.codehaus.grepo.statistics.collection.StatisticsCollectionImpl* holds all collection entries - grepo's *SimpleStatisticsCollectionPrinter* uses a *StatisticsCollection* instance for printing out summary and details. Because grepo stores collection entries in memory by default, there are two properties for the *StatisticsCollectionImpl* which should be configured appropriately to avoid wasting of memory:

- **maxNumberOfRecentStatisticsEntries**: This value controls how many (recent) invocations should be maintained. In the default configuration this value is set to *30*.
- **maxNumberOfTopDurationStatisticsEntries**: This value controls how many (top duration) invocations should be maintained. In the default configuration this value is set to *5*.

Additional functionality

Exporting SimpleStatisticsCollectionPrinter to JMX

We have already used grepo's *SimpleStatisticsCollectionPrinter* in the examples above. This class is appropriately annotated so that it is easy to export instances of this class to a JMX server using springs *MBeanExporter* as follows:

```
<bean
class="org.springframework.jmx.export.MBeanE
xporter">
    <property name="beans">
        <map>
            <entry
key="grepo.statistics:name=StatisticsInforma
tion"
value-ref="statisticsCollectionPrinter" />
        </map>
    </property>
    <property name="assembler">
        <bean
class="org.springframework.jmx.export.assemb
ler.MetadataMBeanInfoAssembler">
            <property
name="attributeSource">
                <bean
class="org.springframework.jmx.export.annota
tion.AnnotationJmxAttributeSource" />
            </property>
        </bean>
    </property>
</bean>
```

Enabling statistics for gquery and gprocedure

The gquery and gprocedure components of the grepo framework can be easily configured for statistics. See the getting started guides for the Generic Query (Hibernte/Jpa) and Generic Procedure components for details.