# 1. Functional testing

## Approach

Functional testing phase was focused on application behavior correctness. Test cases were designed to find run-time errors and memory leaks. Tests were manual.

Tools listed below have been used for investigation:

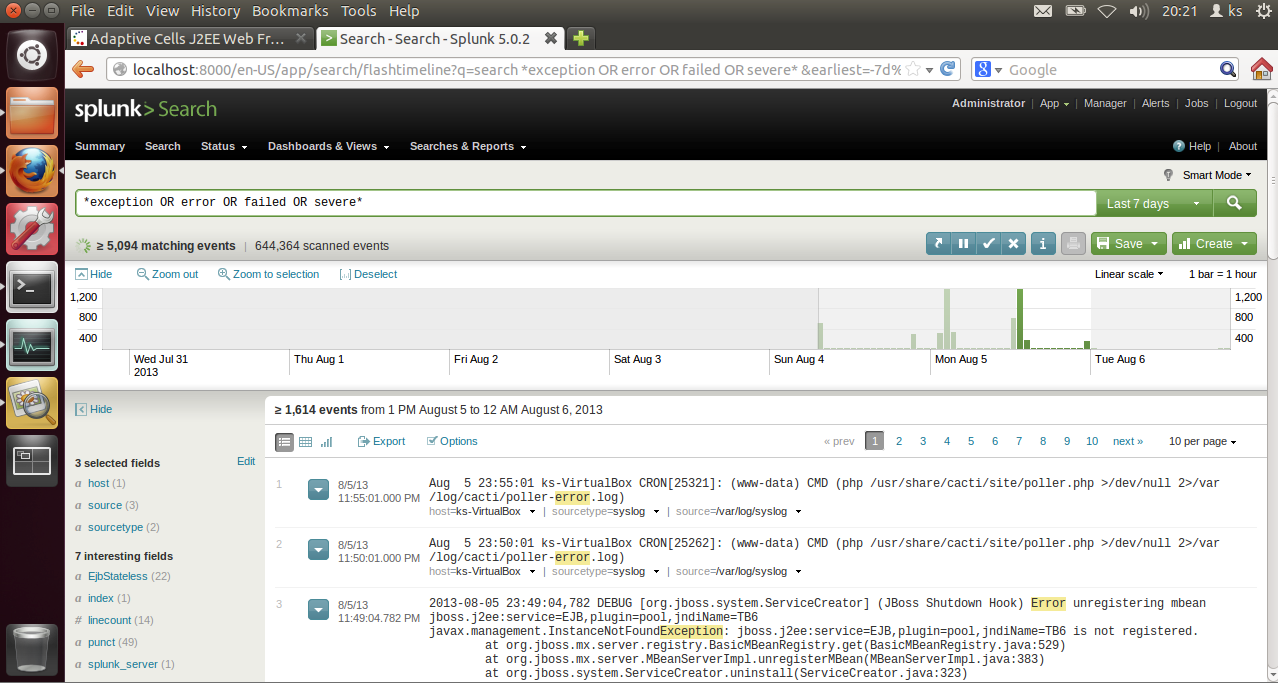
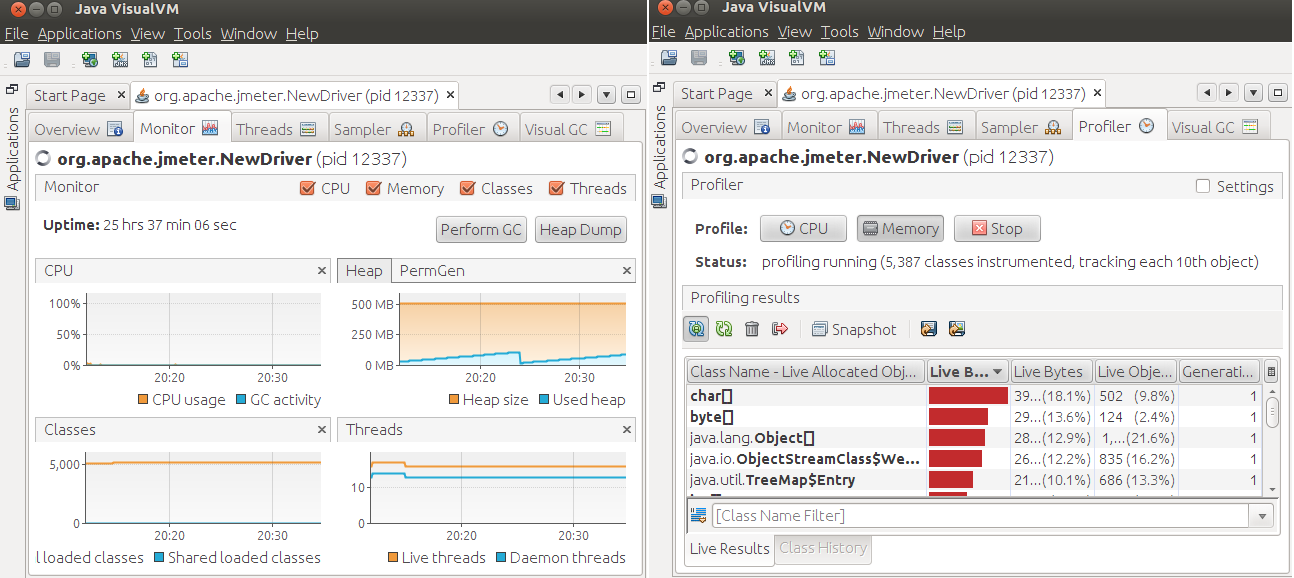
* Splunk - log mining tool for search, analysis and visualization

Figure 1: Splunk query results screenshot.

* Visualvm – JVM monitoring and profiling tool

Figure 2: Visualvm monitoring and profiling views.

Tests cases were executed against every AdaptiveCells/J configuration available (config1...10).

Jboss server and boot logs (JBOSS/server/default/log/server.log, boot.log and access.log) were indexed by Splunk during tests.

For each config, 2 types of test templates were used:

1. run-time exceptions search:

* start Jboss and let it warm up
* start and attach Visualvm
* run config(1...10) once
* repeat every 2 minutes for 3-4 times
* run config(1…10) multiple times
* detach Visualvm
* stop Jboss
* analyse logs using Splunk query that is searching for following keywords in logs:

“exception OR error OR failed OR severe OR ( sourcetype=access\_\* ( 404 OR 500 OR 503 ) )”

1. memory leak recognition:

* start Jboss and let it warm up
* start and attach Visualvm
* start Visualvm memory profiler tracking every object allocation and stack traces
* switch off profiler results automatic refreshing
* kick off garbage collection
* clean all profiler results

\*\*\* at this point object instances are garbage collected and only new allocations will be displayed in profiler results \*\*\*

* execute config(1…10) several times
* kick off garbage collection again
* results contains objects that were not cleaned up properly
* create heap dump
* focus on objects that survived most garbage collections – generations metric in profiler
* for suspicious objects find and investigate stack traces in profiler and find references to instances in heap dump

## Results

Before starting any testing executed ‘run-time exceptions search’ test case to find what exists before application deployment. Tested clean Jboss, with Visualvm attached and with memory profiler running. Identified numerous exceptions thrown by server before application deployment. Highlights:

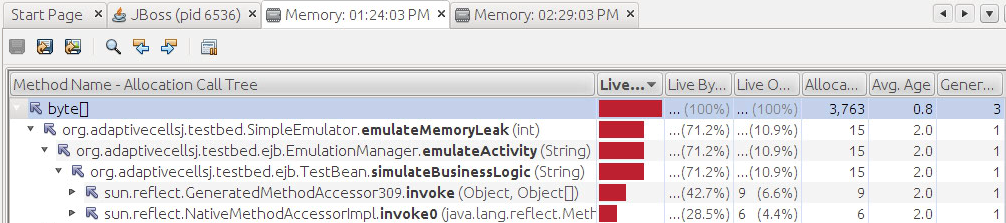
* javax.management.InstanceNotFoundException: jboss.j2ee:service=EJB,plugin=pool,jndiName=TB6 is not registered. thrown by org.jboss.mx.server.registry.BasicMBeanRegistry.get(BasicMBeanRegistry.java:529) – affects all TB (1 to 6) instances
* numerous java.sql.SQLException: Table already exist, java.sql.SQLException: Index already exist, java.sql.SQLException: Violation of unique constraint SYS\_PK\_48: duplicate value(s) for column(s) $$
* java.lang.ClassNotFoundException: org.jboss.mx.server.MBeanServerBuilderImpl Caused by: javax.management.JMRuntimeException: Failed to load MBeanServerBuilder
* java.lang.IllegalArgumentException: Property is not readable: propertyReplace for org.jboss.beans.metadata.plugins.AbstractPropertyMetaData

All of those types of exceptions were investigated for every test case run and excluded from further analysis if no new instance of a given type found.

### Config 1

No application specific run-time exceptions found.

Found memory leak. Identified suspicious object that was the biggest one on the heap and lived for 3 generations: byte[]

Figure 3: Visualvm memory profiler results – allocation call tree.

Suspected method: org.adaptivecellsj.testbed.SimulateEmulator.emulateMemoryLeak(int).

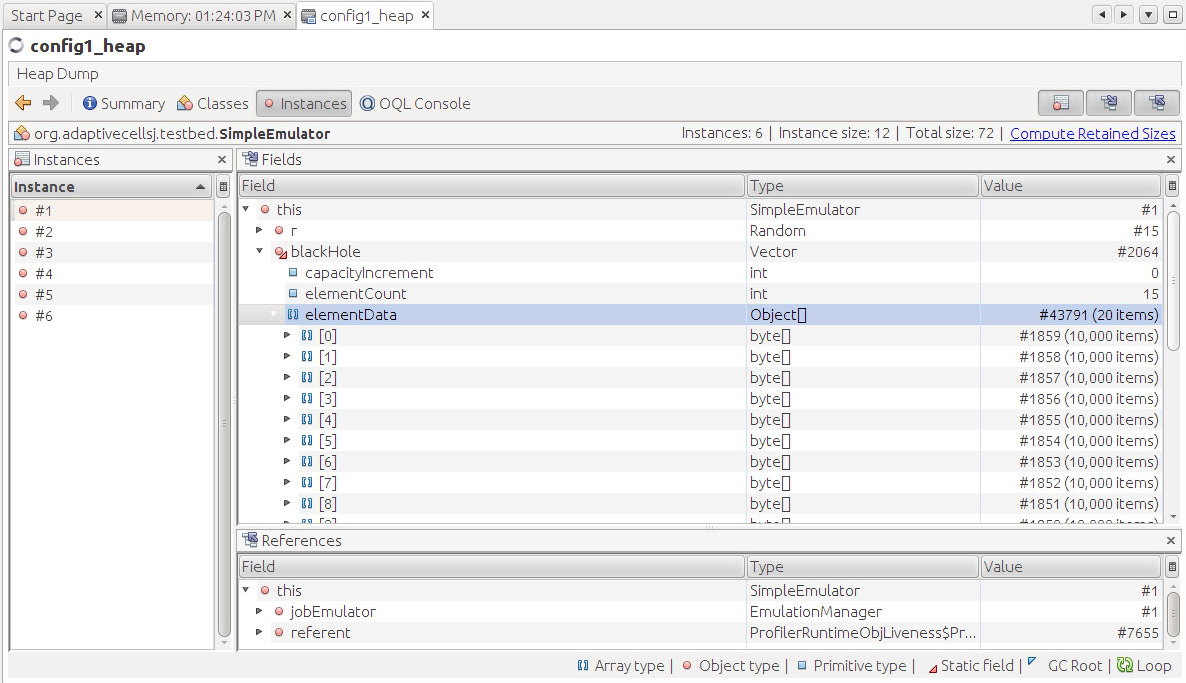
Therefore, checked for SimpleEmulator object references in heap dump. Found 6 instances:

Figure 4: Heap dump view in Visualvm

All of SimpleEmulator instances contains a reference variable to object Vector blackHole that is encapsulating references to several byte[]objects that were not garbage collected properly. Those are the leaked objects.

Exactly the same instances of SimpleEmulator and byte[] objects leaked in Config 2 and Config 7.

### Config 4

Found run-time exception thrown directly in the browser:

The Adaptive Cells EJBs have raised the exception:

null; nested exception is: java.lang.RuntimeException

You might have configured the cells to raise exceptions. This can be done adding an environment entry (namely configXexception) to the deployment descriptors of the cell EJBs.

Found the same exception in Jboss server.log:

TransactionRolledBackException in method: org.adaptivecellsj.testbed.ejb.TestBeanIF.simulateBusinessLogic(java.lang.String) throws java.rmi.RemoteException,java.lang.Exception,

caused by RuntimeException thrown from: org.adaptivecellsj.testbed.SimpleEmulator.emulateException(SimpleEmulator.java:69)

Exception occurrences in logs are aligned with Config 4 execution times from browser.

Exactly the same exception occurred for Config 9.

No memory leaks found.

Summary for all findings:

|  |  |  |
| --- | --- | --- |
|  | Run-time error | Memory leak |
| Config 1 |  | x |
| Config 2 |  | x |
| Config 3 |  |  |
| Config 4 | x |  |
| Config 5 |  |  |
| Config 6 |  |  |
| Config 7 |  | x |
| Config 8 |  |  |
| Config 9 | x |  |
| Config 10 |  |  |

Table 1: Functional testing findings.

Only Configs: 3, 5, 6, 8, 10 will be included in further testing.

Raw data and detailed results from testing can be found:

* clean jboss results: <http://bit.ly/11Ku8kD>
* full results: http://bit.ly/191s1uS