Performance and Status Monitoring of JavaEE Business Applications

Andreas Wälchli
Roadmap

• Project Description
• SOAP WebService
• JavaEE
• JMX
• Project Constraints
• GC Statistics Without Storing Data Points?
• TimerService Monitoring
Project Description

• Customer: ISC-EJPD

• Implement the ApplicationCheck WebService
  → Provide Status Information about the Application

• Configurable, Extendable, Generic
SOAP WebService

• Network Socket on a defined Path & Port
• Client sends a Message and receives a Response (both XML)
• Message and Response Schema defined through a WSDL
ApplicationCheck – WebService

Requirements:

• 1 Predefined “query” message
• Server performs checks and returns a result
• Results can be nested

@javax.ejb.Local
public interface IC徹cker {
    public CheckResponse check(String uid);
}

class CheckResponse implements Serializable {
    private String checkName;
    private String message;
    private String description;
    private String stackTrace;
    private String errorMessage;
    private CheckResult internalResult;
    private List<CheckResponse> subChecks;
}

public enum CheckResult {
    CHECK_OK, CHECK_FAILED;
}
ApplicationCheck – Restrictions

• Responses are binary (CHECK_OK, CHECK_FAILED)
• Raw data is often fuzzy
• Check does not necessarily fail if a sub-check fails

→ Apply a decision function to determine result
Java Enterprise Edition (JavaEE)
Java Management Extension (JMX)

• Provide JVM diagnostic information through JMX Beans
• Specifics are VM dependant
• Remote Monitoring
• Notification Listeners

However:

“This platform extension is only available to the garbage collector implementation that supports this extension.” – JavaDoc

→ Always need to check if specific element actually supports JMX
JMX Bean Example: GarbageCollectorMXBean

java.lang.management.GarbageCollectorMXBean
• getCollectionCount() : long
• getCollectionTime() : long

com.sun.management.GarbageCollectorMXBean
• getCollectionCount() : long
• getCollectionTime() : long
• getLastGcInfo() : GcInfo

GcInfo:
• Timestamp + GC duration
• Detailed before/after data for each VM memory pool:
  EdenSpace, SurvivorSpace 1, SurvivorSpace 2, OldGen, PermGen, CodeGen
Project Constraints: Minimize Everything
GC Statistics Without Storing Data Points?

- 1 GcInfo instance per GC run
- Create statistics from that data

- Sliding average over fixed time period

Problem: Sliding average requires all data points in period

→ Discrete time steps
Discrete Time Steps

• Collect all data points in a short period into a single period object
• This period object holds min/avg/max/count
• Collect a small (known and fixed) number of period object
• Discard period object if too old
Discrete Time Steps – Pros/Cons

+ constant-time data point addition
+ Data points can be discarded
+ Number of period objects fixed and known
+ Small data sets for statistics generation
+ Old period objects can be “merged” to cover larger time periods
+ Allows potentially “infinite” time periods (> 3 Gyr)
– Statistics period varies

→ minimizes “everything”
TimerService

• Call methods on defined schedule
• Sometimes they don’t work
• TimerService only provides basic information
• No direct approach to monitor past invocations

→ Interceptors
Interceptor

- Intercept method calls:
  - Timer methods
  - EJB method calls
  - ...

- Handle Cross-Cutting Concerns

- Configured through configuration, annotations or programatically
TimerService – Invocation Monitoring Strategy

• Intercept all timed method invocations
• Record the invocation
• Check for exceptions

→ Requires Interceptors to be configured in source or during startup