**Author:** Eric Nicosia

**Version:** 15

**Date:** 2013.10.04

**Pôle Architecture technique**

**Tuning des performances**

0

Table of ContenTs

[1 Introduction 3](#_Toc256000000)

[2 Paramétrage des options de la JVM 4](#_Toc256000001)

[3 Paramétrage des options Tomcat 5](#_Toc256000002)

[3.1 Activer la compression des données 6](#_Toc256000003)

[3.2 Modifier le protocole utilisé pour le connecteur HTTP 6](#_Toc256000004)

[3.3 Exemple de paramétrage Tomcat d'un connecteur HTTP avec compression activée et NIO 7](#_Toc256000005)

[4 Annexe 8](#_Toc256000006)

* [Introduction](#scroll-bookmark-1)
* [Paramétrage des options de la JVM](#scroll-bookmark-2)
* [Paramétrage des options Tomcat](#scroll-bookmark-3)
  + [Activer la compression des données](#scroll-bookmark-4)
  + [Modifier le protocole utilisé pour le connecteur HTTP](#scroll-bookmark-5)
  + [Exemple de paramétrage Tomcat d'un connecteur HTTP avec compression activée et NIO](#scroll-bookmark-6)
* [Annexe](#scroll-bookmark-7)

Introduction

Apache Tomcat est un serveur applicatif écrit en Java. Les optimisations de performances se feront donc de deux façons :

* Paramétrage de la JVM pour le lancement de Tomcat
* Paramétrage des options du Tomcat

Paramétrage des options de la JVM

Ne seront abordés ici que les paramètres disponibles pour la JVM de Sun (Oracle), qui est la JVM utilisée à médiapost.  
La JVM dispose d'un nombre élevé de paramètres qui peuvent lui être appliqués et dont vous trouverez la liste ici : <http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>

Les premiers paramètres à considérer concernent la mémoire. Celle-ci doit être configurée en fonction de la consommation mémoire de votre application et du nombre d'utilisateurs simultanés prévus.

Optimiser la gestion mémoire de la JVM implique une bonne connaissance des mécanismes de gestion de la mémoire par la JVM. Ceux-ci sont décrit en détail là : <http://www.oracle.com/technetwork/java/whitepaper-135217.html>

|  |
| --- |
| * Une règle de base à respecter absolument pour avoir des performances optimale : Le total de mémoire pouvant être alloué à la JVM doit être disponible en mémoire vive sur la machine afin d'éviter l'utilisation du swap (écriture sur disque) qui dégraderait les performances de façon drastique * Corolaire de la première règle : Sur les serveurs physiques hébergeant des machines virtuelles, le total de mémoire max configurée pour les JVM hébergés par les VM ne doit pas dépasser la mémoire physique de la machine (dans le cas de VM gérant l'allocation dynamique de mémoire). |

Il s'agit des options suivantes :

* -Xmx 2500m Taille maximale de la pile mémoire configurée à 2.5 Go
* -Xms 2500m Taille initiale de la pile mémoire configurée à 2.5 Go
* -XX:NewSize=768m Taille initiale de la New Generation
* -XX:MaxNewSize=768m Taille maximale de la New Generation
* -XX:PermSize=128m Taille initiale de la Permanent Generation
* -XX:MaxPermSize=256m Taille maximale de la permanent Generation
* -XX:ThreadStackSize=128 Thread Stack Size configuré à 128 Ko

Vient ensuite le paramétrage du Garbage Collector de la JVM.

* -XX:+UseParNewGC Active l'utilisation du parallel garbage collection
* -XX:+UseConcMarkSweepGC
* -XX:+CMSParallelRemarkEnabled
* -XX:CMSInitiatingOccupancyFraction=70
* -XX:ParallelGCThreads=4 - Défini le nombre de thread utilisés durant les phases de parallel Garbage Collect.
* -XX:+DisableExplicitGC - Désactive les appels a System.gc().

Pour suivre l'utilisation du Garbage collector, on peut activer le logging au niveau de la JVM avec les options suivantes. Sans configuration d'un fichier de log, celles-ci seront écrites dans System.err :

* -XX:+PrintGCDetails
* -verbosegc
* -XX:+PrintGCTimeStamps
* -Xloggc:<filename> - Loggue les infos de verbosegc dans un fichier spécifique.
* -XX:-UseGCLogFileRotation - Active la rotation des logs de GC pour limite la taille de celles-ci (Nécessite -Xloggc)
* -XX:NumberOfGClogFiles=2 - Défini le nombre maximal de fichiers de log quand la rotation de fichier est activée. (>=1)
* -XX:GCLogFileSize=10M - Défini la taille maximale d'une log de GC. ( >=8K)

Enfin, il peut être intéressant d'avoir une trace lors d'erreurs OutOfMemory, c'est à dire un dump de la mémoire au moment de l'erreur, afin de pouvoir analyser les objets présents en mémoire à ce moment là et établir un diagnostic.

* *-XX:-HeapDumpOnOutOfMemoryError - Active le dump de la mémoire en cas d'erreur OutOfMemory*
* *-XX:HeapDumpPath=./java\_pid<pid>.hprof - Emplacement ou doivent être stockés les fichiers de dump*

Paramétrage des options Tomcat

Le serveur Tomcat utilise un fichier de paramétrage nommé **server.xml**, présent dans le sous répertoire /conf. C'est dans ce fichier qu'il faudra intervenir pour modifier le paramétrage.

Activer la compression des données

Les navigateurs modernes supportent tous la compression de données (gzip, deflate), ainsi que la plupart des serveurs applicatifs (Tomcat, Jboss, ...)

Il peut être avantageux d'utiliser la compression pour plusieurs raisons :

* Réduire la consommation de la bande passante réseau.
* Réduire le temps de chargement des pages sur le navigateur

|  |
| --- |
| La compression des données ne doit être activée que pour les données de type texte. En effet, les images de type png ou jpeg, ou les fichiers ZIP et PDF sont déjà compressées. L'activation de la compression pour ce type de données n'apporterai aucun gain, voire dégraderai les performances avec un surcout en charge CPU côté serveur et client et un volume de données transmises sur le réseau supérieur au volume sans compression. |

L'activation de la compression des données sur Tomcat s'effectue en ajoutant les paramètres suivants au connecteur http défini dans le fichier server.xml (<Connector port="8080" protocol="HTTP/1.1" ... />)

**compression="on"** : Autorise la compression de données sur le serveur Tomcat

**compressionMinSize=2048** : Seules les données d'un volume supérieur à 2048 octets pourront être compressées.

**noCompressionUserAgents="gozilla, traviata"** : Les réponses pour les navigateurs identifiés comme gozilla ou traviata ne seront pas compressées.

**compressableMimeType="text/plain, text/html, text/xml, text/css, application/javascript, application/x-javascript, text/javascript, text/x-js"** : Seules les réponses du type mime spécifié seront compressées.

Modifier le protocole utilisé pour le connecteur HTTP

Jusqu'à la version 7 (inclus) de Tomcat, l'API pour le protocole par défaut utilisé est le BIO (ou Blocking I/O). Il est basé sur le modèle :une connexion HTTP / un Thread.  
L'API Java NIO (New IO) permet de gérer plusieurs connexions HTTP avec un seul Thread.  
D'autre part, avec le BIO, lorsqu'un thread attend ou envoie des données, il est 'bloqué' jusqu'à la réception ou l'envoi complet des données, et ne peut rien faire d'autre. Un thread utilisant NIO peut en revanche traiter plusieurs requêtes simultanément. Le gain est évident : moins de thread pour traiter un même nombre de requêtes, moins d'occupation mémoire et de temps CPU et au final une meilleure réactivité et tenue en charge du serveur.

La modification du protocole se fait dans le fichier server.xml, en modifiant l'attribut protocol du tag connector :

**protocol="org.apache.coyote.http11.Http11NioProtocol"**

Exemple de paramétrage Tomcat d'un connecteur HTTP avec compression activée et NIO

|  |
| --- |
| <Connector port="8080" protocol="org.apache.coyote.http11.Http11NioProtocol"  connectionTimeout="20000"  redirectPort="8443"   URIEncoding="UTF-8"  compression="on"   compressionMinSize="2048"   noCompressionUserAgents="gozilla, traviata"   compressableMimeType="text/plain, text/html, text/xml, text/css, application/javascript, application/x-javascript, text/javascript, text/x-js, application/xhtml+xml" /> |

Annexe

Plus d'informations sont disponibles concernant les différences et les avantages de chaque type de connecteurs HTTP ici : <http://tomcat.apache.org/tomcat-7.0-doc/config/http.html>. Cette page liste également l'ensemble des paramètres disponibles pour chaque type de connecteur.