

Hitachi Application Server

サイジングガイド [UNIX]

性能要件を満たすためのマシンリソースおよびマシン台数の算出方法

2016. **10**
October

はじめに

本書は、マニュアル「Hitachi Application Server V10 ユーザーズガイド (UNIX(R)用)」の「1.1 マニュアルの読み方について」で定義している「Web フロントシステム」の、提案・設計フェーズで使用するドキュメントとして次の基準に基づいて記述しています。

1. 対象とする読者

Web フロントシステムを提案・設計する立場にあるプロジェクト管理者を対象としています。本書によって、「Web フロントシステム」のサイジングを実施できます。

2. 対象とする製品

Hitachi Application Server 10-11

■商標類

- ・ HITACHI は、株式会社 日立製作所の商標または登録商標です。
- ・ AMD は、Advanced Micro Devices, Inc.の商標です。
- ・ IBM、AIX は、世界の多くの国で登録された International Business Machines Corporation の商標です。
- ・ Intel は、アメリカ合衆国およびその他の国における Intel Corporation の商標です。
- ・ Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
- ・ Microsoft および Excel、Microsoft Office は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
- ・ Oracle と Java は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。
- ・ This product includes software developed by Andy Clark.
- ・ This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (<http://relaxngcc.sf.net/>).
- ・ This product includes software developed by IAIK of Graz University of Technology.
- ・ Red Hat は、米国およびその他の国で Red Hat, Inc. の登録商標もしくは商標です。
- ・ RSA および BSAFE は、米国 EMC コーポレーションの米国およびその他の国における商標または登録商標です。
- ・ UNIX は、The Open Group の米国ならびに他の国における登録商標です。
- ・ 本製品は、米国 EMC コーポレーションの RSA BSAFE(R)ソフトウェアを搭載しています。
- ・ その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

■マイクロソフト製品のスクリーンショットの使用について

Microsoft Corporation のガイドラインに従って画面写真を使用しています。

■ 製品の表記

本書では、製品名を次のように表記しています。

表記	製品名
Application Server	Hitachi Application Server
Microsoft Office Excel	Microsoft® Office Excel

■ 英略語の表記

本書では、英略語を次のように表記しています。

表記	英字での表記
EAR	Enterprise Archive
EJB	Enterprise JavaBeans™
Java	Java™
JavaEE	Java™ Platform, Enterprise Edition
JavaVM	Java™ Virtual Machine
WAR	Web Archive

■ 発行元

株式会社日立製作所 ICT 事業統括本部 サービスプラットフォーム事業本部

All Rights Reserved. Copyright (C) 2014, 2016, Hitachi, Ltd.

サイジングガイド

性能要件を満たすためのマシンリソースおよびマシン台数の算出方法

目次

1 サイジング概要	1
1.1 サイジングとは.....	2
1.2 要求リソースの考え方.....	3
2 マシンサイジング	5
2.1 システム全体に必要なマシンリソースの算出.....	6
2.1.1 CPU コア数の算出.....	6
2.1.2 メモリーサイズの算出.....	8
2.1.3 ディスク容量の算出.....	18
2.2 マシン台数の算出.....	21
2.2.1 マシンスペックの決定.....	21
2.2.2 マシン台数の算出.....	21
2.2.3 マシン 1 台当たりの Application Server が使用するリソース.....	22
3 処理性能の求め方	23
3.1 処理性能の概算.....	24
3.2 処理性能の測定.....	26
3.2.1 実行 CPU 時間.....	26
3.2.2 内部保留時間.....	28
3.2.3 1 スレッド使用 Survivor 領域サイズ、共通使用 Survivor 領域サイズ.....	29
3.2.4 アプリケーション使用 Tenured 領域サイズ.....	32
3.2.5 1 セッション当たりの使用メモリーサイズ.....	34

1 サイジング概要

この章ではサイジングの概要について説明します。

本章の構成

- 1.1 サイジングとは
- 1.2 要求リソースの考え方

1.1 サイジングとは

業務のシステム要件を満たすマシン構成を決定するプロセスを、サイジングと呼びます。言い換えると、システム要件のうち、性能要件と信頼性要件をマシンリソースへ換算するプロセスです。

本書では性能要件を基に **Application Server** が必要とするマシンリソース(要求リソース)の算出方法と、要求リソースを実現するマシン構成(マシンスペックとマシン台数)の算出方法を示します。この算出結果と信頼性要件を基に実際のマシン台数を調節してください。

ここでサイジング対象のマシンリソースは、CPU コア数、物理メモリーサイズ、ディスクサイズを指します。

1.2 要求リソースの考え方

要求リソースは、スケールアウトして分割できるリソース(分割可能リソース)とスケールアウトしても分割できないリソース(固定リソース)から構成されます。

例えば、ディスクサイズを考えたとき、Application Server のログ容量はマシン台数を増やすと1台当たりの処理件数が減少するので分割可能リソースとなります。また、Application Server のインストール容量はすべてのマシンに必要なので固定リソースとなります。

分割可能リソース・固定リソースのイメージを次に示します。

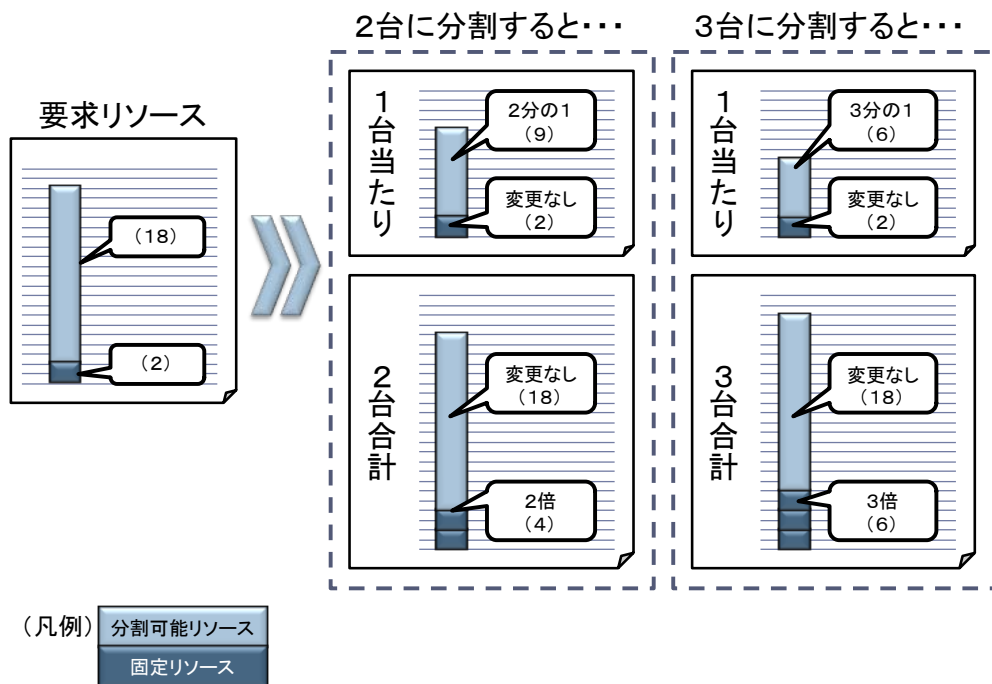


図 1.2-1 分割可能リソース・固定リソースのイメージ

1台のマシンの、マシンスペックが固定リソースを上回った分だけ分割可能リソースを負担できます。例えば、マシンスペックが11で、固定リソースが2とすると、1台当たり9ずつ分割可能リソースを負担できます。

分割可能リソース・固定リソースを勘案することで、マシン台数（スケールアウト・水平スケーリング）とマシンスペック（スケールアップ・垂直スケーリング）を適切に組み合わせるマシン構成を実現できます。

マシン構成のイメージを次に示します。

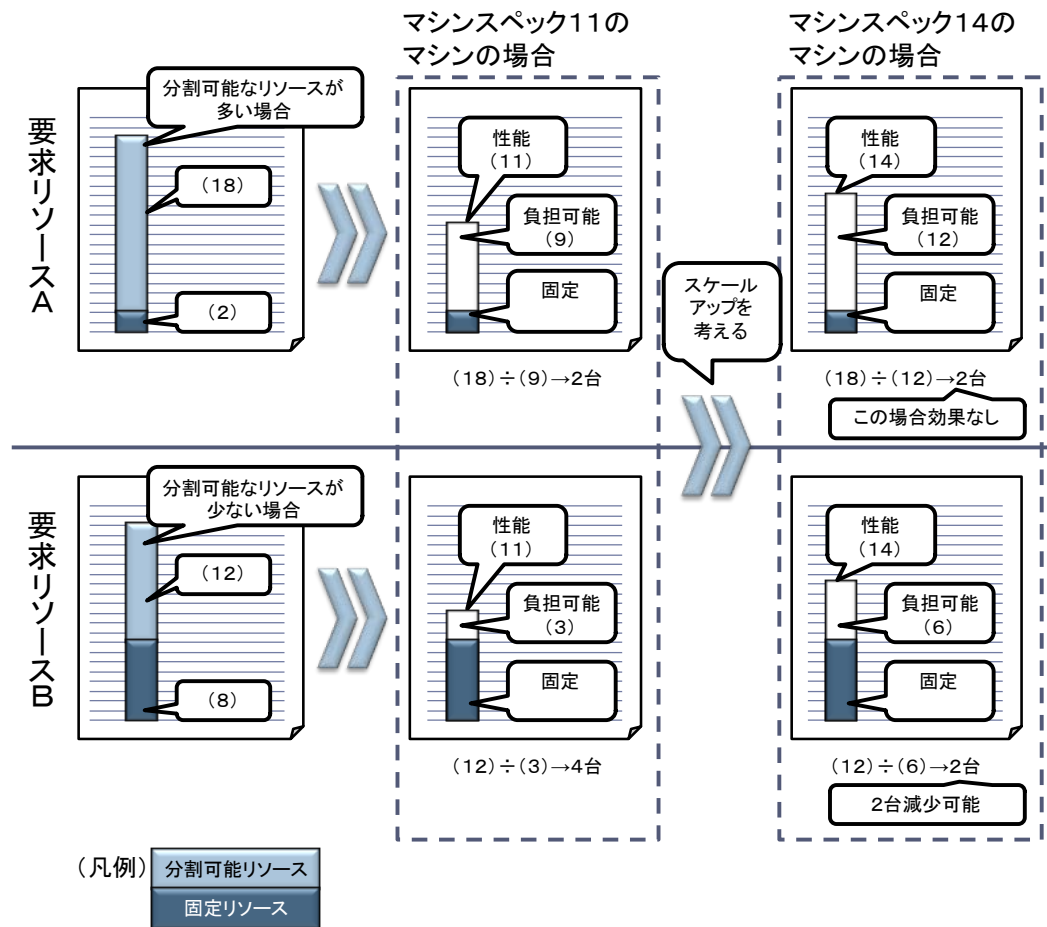


図 1.2-2 マシン構成のイメージ

2 マシンサイジング

この章では、マシンサイジングについて説明します。

本章の構成

- 2.1 システム全体に必要なマシンリソースの算出
- 2.2 マシン台数の算出

2.1 システム全体で必要なマシンリソースの算出

Application Server がシステム全体で必要とするマシンリソースの算出方法を示します。またマシン台数の算出のために、マシンリソースを、マシン 1 台ごとに分割できる分割可能リソースと、マシンごとに分割できない固定リソースに分けて算出する方法についても示します。

2.1.1 CPU コア数の算出

(1) 算出に必要な情報

CPU コア数の算出に必要な情報を次に示します。

システム要件はお客さまに提示してもらう情報です。処理性能はアプリケーションやマシンの特徴に依存するため、基礎値を用いた概算値またはプロトタイプの実測値を基に決定します。

#	カテゴリ	要素名	備考
1	システム要件	1 秒当たりのリクエスト数 [件/秒]	1 秒当たりにシステムが受け付けるリクエストの数です。
2	処理性能	実行 CPU 時間[秒]	1 リクエスト当たりの CPU 時間です。
3		CPU 使用率[%]	参考値 : 70
4		安全係数	参考値 : 1.3

(2) 算出方法

1 秒当たりのリクエスト数から、必要な CPU コア数を見積もります。1 秒当たりのリクエスト数が 20 件のシステムを例に、基本的な考え方を説明します。

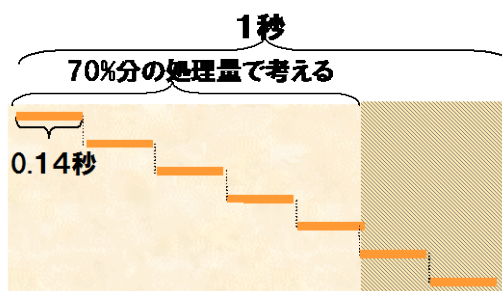


図 2.1-1 1 個の CPU で処理できるリクエスト数

まず、1 個の CPU で 1 秒に処理できるリクエスト数を求めます。リクエスト 1 件の処理に 0.14 秒掛かる CPU の場合(実行 CPU 時間=0.14 秒)、CPU1 個で 1 秒に処理できるリクエスト数は $1/0.14 \text{ 秒} \approx 7.1$ 件であり、1 秒に約 7 件です。しかし、これは CPU を 100% 使った場合に処理できる件数であり、実際には 100%稼働させると CPU ネックになってしまいます。CPU 使用率 70%で処理できるリクエスト数は、 $(1/0.14 \text{ 秒}) \times 0.7 = 5$ 件であり、1 秒に 5 件です。

つまり、1個のCPUで1秒に処理できるリクエスト数は次の式から求められます。

$$\text{1個のCPUで1秒に処理できるリクエスト数[件]} = \frac{1[\text{秒}]}{\text{実行CPU時間[秒]}} \times \frac{\text{CPU使用率}[\%]}{100}$$

処理しなければならない1秒当たりのリクエスト数を、上記で求めた1個のCPUで1秒に処理できるリクエスト数で割ると、必要なCPUコア数を算出できます。

$$\begin{aligned} \text{CPUコア数[個]} &= \frac{\text{1秒当たりのリクエスト数[件/秒]}}{\text{1個のCPUで1秒に処理できるリクエスト数[件]} \times \text{安全係数}} \\ &= \frac{\text{1秒当たりのリクエスト数[件/秒]} \times \text{実行CPU時間[秒]} \times \text{安全係数}}{\frac{\text{CPU使用率}[\%]}{100}} \end{aligned}$$

(3) 分割可能リソース・固定リソースの算出方法

1秒当たりのリクエスト数はマシンごとに分割されるため、CPUコア数はすべて分割可能リソースです。

$$\text{分割可能リソース} = \frac{\text{1秒当たりのリクエスト数[件/秒]} \times \text{実行CPU時間[秒]} \times \text{安全係数}}{\frac{\text{CPU使用率}[\%]}{100}}$$

$$\text{固定リソース} = 0$$

2.1.2 メモリーサイズの算出

(1) 算出に必要な情報

メモリーサイズの算出に必要な情報を次に示します。

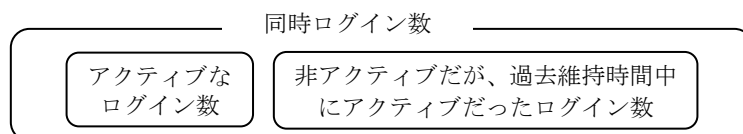
システム要件はお客さまに提示してもらう情報です。処理性能はアプリケーションやマシンの特徴に依存するため、基礎値を用いた概算値またはプロトタイプの実測値を基に決定します。

#	カテゴリ	要素名	備考
1	システム要件	1 秒当たりのリクエスト数 [件/秒]	1 秒当たりにシステムが受け付けるリクエストの数です。
2		同時ログイン数[人]	同時にシステムにログインできる利用者の数です。ログアウトが正常に行われていない場合に維持されるログイン数も含めて見積もります。
3	処理性能	内部保留時間[秒]	リクエストがアプリケーションサーバ内に入ってから出るまで、データベースサーバとのやり取りも含めた処理時間です。
4		1 スレッド使用 Survivor 領域サイズ[MB]	リクエスト・レスポンス処理で、1 スレッドが使用する短寿命なオブジェクトのサイズです。
5		共通使用 Survivor 領域サイズ[MB]	リクエスト・レスポンス処理で、全スレッドが共通で使用する短寿命なオブジェクトのサイズです。
6		アプリケーション使用 Tenured 領域サイズ[MB]	複数リクエストにまたがって使用する長寿命なオブジェクトのサイズです。
7		1 セッション当たりの使用メモリーサイズ[MB]	1 セッションで保有するオブジェクトのサイズです。
8		ノード数	実行系を構成するノードの数です。
9		安全係数	参考値：1.3



参考 同時ログイン数の考え方

Web ブラウザーの×ボタンをクリックするなど、ログアウトが正常に行われていない場合、そのログイン状態が維持されます（維持時間(=セッションタイムアウト時間)はアプリケーションに設定。デフォルトでは 30 分)。維持時間中はメモリーを使用し続けるため、同時ログイン数としてカウントします。



(2) 算出対象

各プロセスが使用するメモリーサイズを合計します。対象となるプロセスを次に示します。

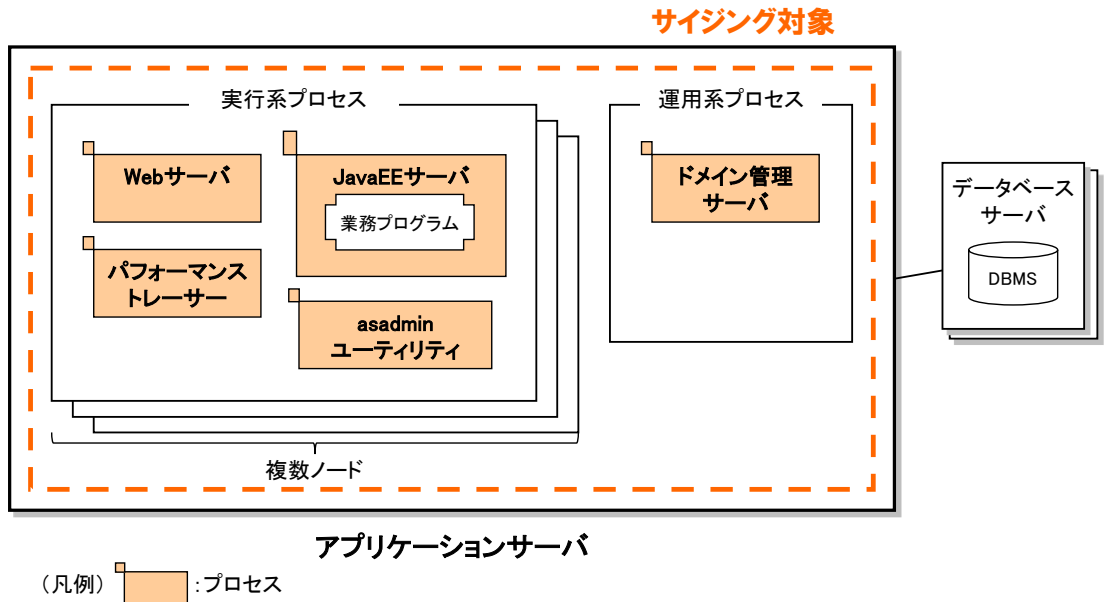


図 2.1-2 メモリーサイズのサイジング対象

各プロセスが使用するメモリーサイズは次のとおりです。

- 実行系プロセス（固定リソースはノード数分必要）
 - ・ パフォーマンストレーサー：20MB
 - ・ Webサーバ：100MB
 - ・ JavaEEサーバ：アプリケーションによって大きく変動
 - ・ asadminユーティリティ：アプリケーションによって大きく変動
- 運用系プロセス
 - ・ ドメイン管理サーバ：820MB

JavaEEサーバおよび asadminユーティリティ以外のプロセスのメモリーサイズは、およその値が想定できるため、ここでは、アプリケーションによって大きく変動するJavaEEサーバおよび asadminユーティリティのメモリーを見積もります。

また、実行系プロセスを複数ノード起動する場合、それらのプロセスの固定リソースはノード数分必要です。ドメイン管理サーバについては、複数ノードを起動することはありません。

(3) Java EE サーバのメモリー算出方法

JavaEE サーバのメモリーは、Java ヒープと Explicit ヒープと Metaspace を見積もることで決定します*。Java ヒープは必要な Survivor 領域・Tenured 領域を確保してメモリー不足にならないように見積もります。Explicit ヒープはセッション情報をすべて格納して FullGC を発生させないように見積もります。Metaspace は JavaEE サーバにロードされるクラスファイルの合計サイズを確保してメモリー不足にならないように見積もります。

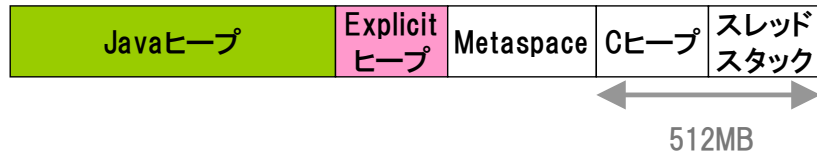


図 2.1-3 JavaEE サーバのメモリー構成

注※ アプリケーションからネイティブライブラリーを使用する場合やスレッドを生成する場合は、別途 C ヒープ、スレッドスタックの見積もりが必要です。

■Java ヒープサイズの見積もり

Java ヒープの構成を次に示します。

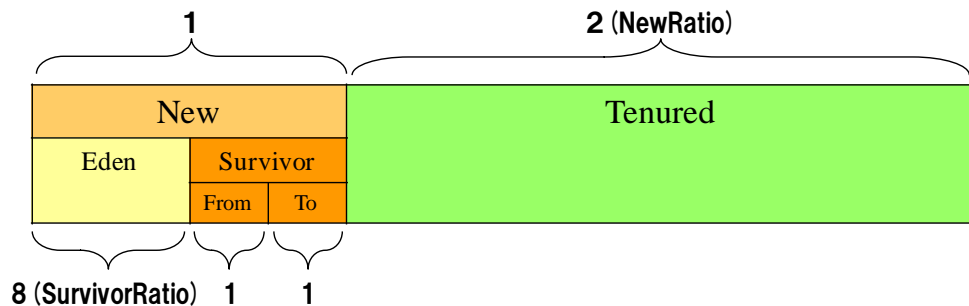


図 2.1-4 Java ヒープの構成

Java ヒープは大きく New 領域と Tenured 領域から構成されていて、それぞれの領域をどれだけ必要とするかは、業務によって異なります。そのため、次の 2 つの観点からそれぞれ必要な Java ヒープサイズを見積もり、大きい方を採用します。

1. New 領域から Java ヒープサイズを見積もる
2. Tenured 領域から Java ヒープサイズを見積もる

1. New 領域から Java ヒープサイズを見積もる

New 領域はリクエスト・レスポンス処理に使用する短寿命なオブジェクトを扱う領域です。この短寿命なオブジェクトの量が Survivor 領域 To 空間の 1/2 を超えてしまうと、超えた分のオブジェクトが Tenured 領域に移動し、FullGC 発生 の 要因 になります。そのため、短寿命なオブジェクトが使用する Survivor 領域サイズ(使用 Survivor 領域サイ

ズ)を基点に Java ヒープサイズを見積もる必要があります。

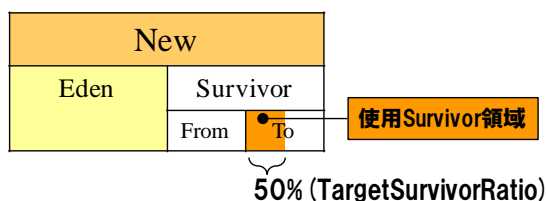


図 2.1-5 使用 Survivor 領域サイズ

計算式は次のようになります。

Java ヒープサイズ[MB]=使用 Survivor 領域サイズ×60※×安全係数

注※ $60 = 2 \times 10 \times 3 = \frac{100}{\text{TargetSurvivorRatio}} \times (\text{SurvivorRatio} + 2) \times (\text{NewRatio} + 1)$

ここで使用 Survivor 領域サイズと多重度の関係をグラフにすると次のようになります。

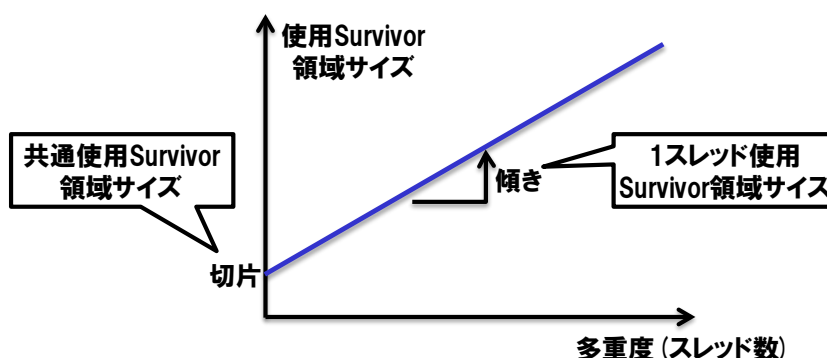


図 2.1-6 使用 Survivor 領域サイズと多重度の関係

1スレッド使用 Survivor 領域サイズはJavaEEサーバの1スレッドが使用するオブジェクトのサイズ、共通使用 Survivor 領域サイズは全スレッドが共通で使用するオブジェクトのサイズです。よって、使用 Survivor 領域サイズは次の式で表されます。

使用 Survivor 領域サイズ[MB]

=1スレッド使用 Survivor 領域サイズ×多重度※+共通使用 Survivor 領域サイズ

注※ 多重度=1秒当たりのリクエスト数[件/秒]×内部保留時間[秒]

以上より、Java ヒープサイズの計算式は次のようになります。

Java ヒープサイズ[MB]

= (1スレッド使用 Survivor 領域サイズ
× 1秒当たりのリクエスト数[件/秒]×内部保留時間[秒]
+ 共通使用 Survivor 領域サイズ)

×60×安全係数

2. Tenured 領域から Java ヒープサイズを見積もる

Tenured 領域は複数リクエストにまたがって使用する長寿命なオブジェクトを扱う領域です(ただし、セッション情報は Explicit ヒープで扱うためここでは除きます)。長寿命なオブジェクトにはアプリケーションが使用するものと、Application Server 自身が使用するもの(固定リソース)とがあり、その合計が Tenured 領域の 1/2 を超えてしまうと、FullGC が発生する可能性が高くなります。そのため、アプリケーションが使用する Tenured 領域サイズ(アプリケーション使用 Tenured 領域サイズ)を基点に Java ヒープサイズを見積もる必要があります。

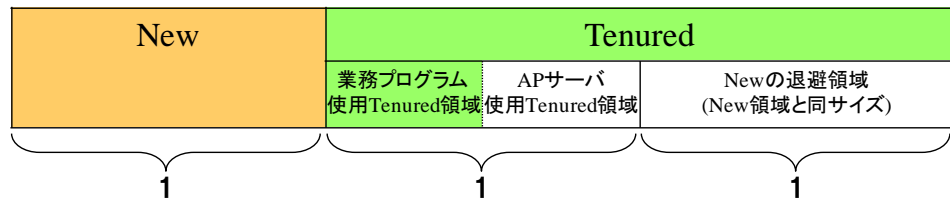


図 2.1-7 アプリケーション使用 Tenured 領域サイズ

計算式は次のようになります。

Java ヒープサイズ[MB]

=アプリケーション使用 Tenured 領域サイズ×3^{*1}×安全係数+300^{*2}×3^{*1}

注※1 3=NewRatio+1

注※2 300=Application Server 使用 Tenured 領域サイズ[MB]

■Explicit ヒープサイズの見積もり

Explicit ヒープは、長寿命だがある期間を経て破棄されるオブジェクトを扱う領域です。そのようなオブジェクトにはアプリケーションで使用するセッション情報と Application Server 自身が使用するものがあり、その合計が Explicit ヒープサイズを超えてしまうと、あふれたオブジェクトが Tenured 領域へ移動し FullGC 発生の要因になります。

セッション情報は、1 セッション当たりの使用メモリーが同時ログイン数分必要です。ここで同時ログイン数はマシン 1 台当たりの数を使用します。Application Server が使用する Explicit ヒープサイズ(Application Server 使用 Explicit ヒープサイズ)は Survivor 領域サイズです。

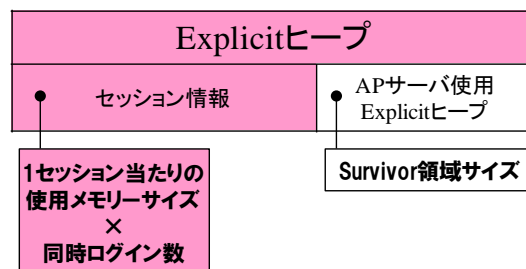


図 2.1-8 Explicit ヒープサイズ

セッション情報サイズ、Application Server 使用 Explicit ヒープサイズの計算式は次のようになります。

セッション情報サイズ[MB]
= $(1 \text{ セッション当たりの使用メモリーサイズ} + 0.0156^{※}) \times \text{同時ログイン数} \times \text{安全係数}$
注※ 0.0156=Explicit ヒープの最小単位[16KB]

Application Server 使用 Explicit ヒープサイズ[MB]
= $(\text{Java ヒープサイズ} / 3^{※1} / 5^{※2})^{※3}$
注※1 $3 = \text{NewRatio} + 1$
注※2 $5 = (\text{SurvivorRatio} + 2) / 2$
注※3 $(\text{Java ヒープサイズ} / 3 / 5) = \text{Survivor 領域サイズ}$

以上より、Explicit ヒープサイズの計算式は次のようになります。

Explicit ヒープサイズ[MB]
=セッション情報サイズ[MB]+Application Server 使用 Explicit ヒープサイズ[MB]
= $(1 \text{ セッション当たりの使用メモリーサイズ} + 0.0156) \times \text{同時ログイン数} \times \text{安全係数}$
+ $(\text{Java ヒープサイズ} / 3 / 5)$

■Metaspace の見積もり

Metaspace でのメモリー使用量は、大体、Java EE サーバにロードされるクラスファイルの合計サイズになります。多くの場合、すべてのクラスがロードされることがないため、検証フェーズですべての業務を実施し、足りなければ増やすという方法で必要な Metaspace のサイズを見積もることができます。

検証フェーズで実測を行うことができない、または事前に Metaspace のサイズを見積もる必要がある場合は、すべてのクラスがロードされた場合の合計サイズを見積もります。ただし、上述の通り、実際はすべてのクラスがロードされることがないため、サイズを大きく見積もることになります。

アプリケーションサーバの場合、次に示すクラスファイルのサイズの合計からすべてのクラスがロードされた場合のサイズを見積もることができます。

1. WEB-INF/classes 以下のすべてのクラスファイル
2. WEB-INF/lib 以下の JAR ファイルに含まれる、すべてのクラスファイル
3. JSP コンパイル結果として生成された、すべてのクラスファイル
4. EJB-JAR に含まれるすべてのクラスファイル
5. 上記以外のアプリケーションに含まれるすべてのクラスファイル

6. `asadmin` コマンドの `add-library` サブコマンドや `deploy` サブコマンドの `--libraries` 引数で追加したすべてのクラスファイル
7. リソースアダプターに含まれるクラスファイル
8. コンテナが作成するクラスファイル
アプリケーション開始直後の `Metaspace` - アプリケーション登録前の `Metaspace` - (1~5 の総和)。
実際に `Java EE` サーバを起動し、`Metaspace` を確認して算出してください。
9. アプリケーションサーバ提供のクラスファイル(システムクラスファイル) : 約 200MB
10. JDK 提供のクラスファイル : 約 110MB

(4) `asadmin` ユーティリティのメモリー算出方法

`asadmin` ユーティリティコマンドの最大メモリー使用量を算出します。算出した値は `HJES_ASADMIN_JVM_OPTIONS` の `-Xmx` オプションで指定します。

■算出に必要な情報

#		要素名	備考
1	システム要件	アプリケーション関連ファイルの総サイズ[MB]	Java EE サーバにデプロイするアプリケーションに関するファイルの総サイズです。
2	定数情報	ファイル変換係数 ^{※1}	参考値：10
3		最小メモリー[MB]	参考値：25

注※1

アプリケーションのデプロイ時に、アプリケーション関連ファイルがドメイン管理サーバから `Java EE` サーバに送信されます。送信時に行われるアプリケーションファイルの変換処理で生じるファイルサイズの増加分を、ファイル変換係数としています。

■アプリケーション関連ファイルの総サイズの算出

アプリケーション関連ファイルとは、ドメイン管理サーバに展開されたアプリケーションのことを指します。`asadmin` ユーティリティの最大メモリー使用量を算出する際は、ドメイン管理サーバを実行しているマシンにある、以下の 5 つのディレクトリーサイズの合計値を、アプリケーション関連ファイルの総サイズとして使用します。

<アプリケーションサーバのインストールディレクトリー>/`javaee/glassfish/domains/`
<ドメイン名> 以下の

- `applications`^{※2※3}
- `config`
- `docroot`
- `generated`
- `lib`

注※2

`applications` ディレクトリー直下の `__internal` ディレクトリーは、算出対象から除外してください。このディレクトリーは `Java EE` サーバに送信されません。

注※3

Java EE サーバごとにデプロイするアプリケーションが異なる場合は Java EE サーバごとに算出し、最も大きな値を使用します。以下に例を示します。

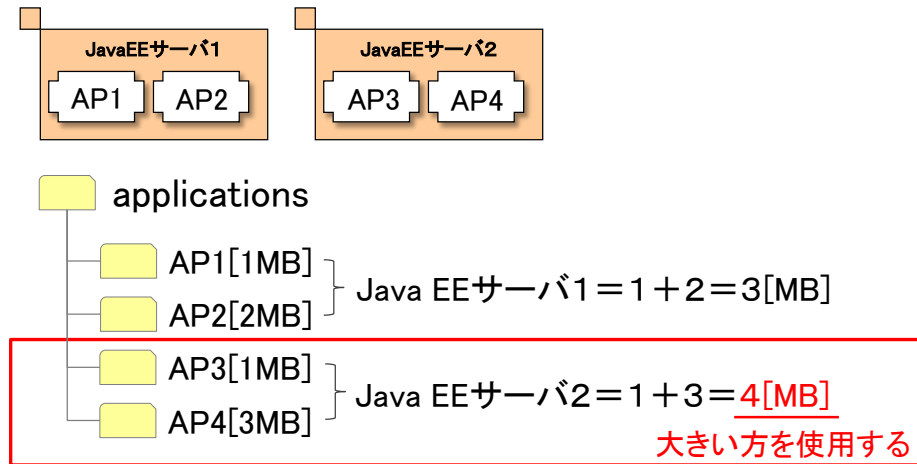


図 2.1-9 applications ディレクトリーの算出

■最大メモリー使用量の算出

asadmin ユーティリティの最大メモリー使用量は次の式から算出します。

最大メモリー使用量[MB]
=アプリケーション関連ファイルの総サイズ[MB]
×ファイル変換係数
+最小メモリー使用量[MB]

(5) 分割可能リソース・固定リソースの算出方法

JavaEE サーバ以外のプロセスのメモリーサイズはすべて固定リソースです。また、JavaEEサーバのうちMetaspace、Cヒープ、スレッドスタックサイズはすべて固定リソースです。Java ヒープ、Explicit ヒープには分割可能リソースと固定リソースがあります。

分割可能リソース

=Java ヒープの分割可能リソース
+ Explicit ヒープの分割可能リソース

固定リソース

= (Web サーバが使用するメモリーサイズ
+ パフォーマンストレーサーが使用するメモリーサイズ
+ Metaspace サイズ
+ C ヒープサイズ+スレッドスタックサイズ
+ Java ヒープの固定リソース
+ Explicit ヒープの固定リソース) ×ノード数
+ ドメイン管理サーバが使用するメモリーサイズ
+ asadmin ユーティリティが使用するメモリーサイズ

■Java ヒープの分割可能リソースと固定リソース

1. New 領域から見積もった Java ヒープの分割可能リソースと固定リソース

1 秒当たりのリクエスト数はマシンごとに分割されるため、1 秒当たりのリクエスト数と乗除されている項が分割可能リソースで、それ以外の項が固定リソースです。

Java ヒープの分割可能リソース

=1 スレッド使用 Survivor 領域サイズ
×1 秒当たりのリクエスト数[件/秒]×内部保留時間[秒]
×60×安全係数

Java ヒープの固定リソース

=共通使用 Survivor 領域サイズ×60×安全係数

ただし、マシン一台当たり最低でも 1 多重を処理する Java ヒープサイズは必要なので、それを最低必要リソースとします。

Java ヒープの最低必要リソース

=(1 スレッド使用 Survivor 領域サイズ×1+共通使用 Survivor 領域サイズ)×60×安全係数

2. Tenured 領域から見積もった Java ヒープの分割可能リソースと固定リソース

Tenured 領域から見積もった Java ヒープサイズはマシンごとに分割される要素がないため、すべて固定リソースです。

Java ヒープの分割可能リソース

=0

Java ヒープの固定リソース

=アプリケーション使用 Tenured 領域サイズ×3×安全係数+300×3

■ Explicit ヒープの分割可能リソースと固定リソース

同時ログイン数はマシンごとに分割されるため、同時ログイン数と乗除されている項が分割可能リソースで、それ以外の項が固定リソースです。

Explicit ヒープの分割可能リソース

= $(1 \text{ セッション当たりの使用メモリーサイズ} + 0.0156) \times \text{同時ログイン数} \times \text{安全係数}$
 +(Java ヒープサイズの分割可能リソース/3/5)

Explicit ヒープの固定リソース

= $(\text{Java ヒープサイズの固定リソース} / 3 / 5)$

ただし、最低でも 1 ログインを処理する Explicit ヒープサイズは必要なので、それを最低必要リソースとします。

Explicit ヒープの最低必要リソース

= $(1 \text{ セッション当たりの使用メモリーサイズ} + 0.0156) \times 1 \times \text{安全係数}$
 +(Java ヒープサイズ/3/5)

2.1.3 ディスク容量の算出

(1) 算出に必要な情報

ディスク容量の算出に必要な情報を次に示します。

システム要件はお客さまに提示してもらう情報です。処理性能はアプリケーションやマシンの特徴に依存するため、基礎値を用いた概算値またはプロトタイプの実測値を基に決定します。

#		要素名	備考
1	システム要件	1 日当たりのリクエスト数 [件/日]	1 日当たりにシステムが受け付けるリクエストの数です。
2		稼働情報保有日数[日]	Application Server が出力する稼働情報を保有する日数です。
3	処理性能	アプリケーションの総容量 [MB]	JavaEE サーバに配置するアプリケーション(EAR ファイルまたは WAR ファイル)の、アーカイブ展開時の総容量です。
4		ノード数	実行系を構成するノードの数です。

(2) 算出方法

ディスク容量は、次の式で計算されます。

$$\text{ディスク容量} = \text{インストール容量} + \text{ログ容量} + \text{ワーク容量} + \text{ダンプ容量}$$

計算式中のインストール容量、ログ容量、ワーク容量、ダンプ容量は次のとおりです。実行系プロセスを複数ノード起動する場合、インストール容量以外の固定リソースをノード数分用意します。

#		説明	容量
1	インストール容量	Application Server をインストールするための容量です。	400MB
2	ログ容量	Application Server が出力する、ログファイル・稼働情報を格納するための容量です。	下記ログ容量の計算式を参照
3	ワーク容量	JavaEE サーバが動作する上で必要な作業領域です。主にアプリケーションを格納するために使用されます。	下記ワーク容量の計算式を参照
4	ダンプ容量	障害時に OS が出力するダンプファイルを格納するための容量です。	マシン 1 台当たりの JavaEE サーバが使用するメモリーサイズ[MB]

ログ容量の計算式を次に示します。

$$\begin{aligned} & \text{ログ容量} \\ & = ((0.001[\text{MB}] \times 1 \text{ 日当たりのリクエスト数}^{\ast 1} + 8[\text{MB}]^{\ast 2}) \times (\text{稼働情報保有日数} + 1)^{\ast 3} \\ & \quad + 980\text{MB}^{\ast 4}) \times 2^{\ast 5} \end{aligned}$$

注※1 0.001[MB]×1日当たりのリクエスト数：Webサーバの1日当たりの稼働情報

注※2 8：JavaEEサーバの1日当たりの稼働情報

注※3 稼働情報保有日数+1：稼働情報ファイルの保有数

注※4 980MB：ログファイルの容量

注※5 2：障害時の情報収集用

ワーク容量の計算式を次に示します。

$$\text{ワーク容量}[\text{MB}] = \text{アプリケーションの総容量}[\text{MB}] \times 4$$

(3) 分割可能リソース・固定リソースの算出方法

ログ容量以外はすべて固定リソースです。ログ容量には分割可能リソースと固定リソースがあります。

分割可能リソース =ログ容量の分割可能リソース 固定リソース =インストール容量 + (ワーク容量 + ダンプ容量 + ログ容量の固定リソース) × ノード数

■ ログ容量の分割可能リソースと固定リソース

1日当たりのリクエスト数はマシンごとに分割されるため、1日当たりのリクエスト数と乗除されている項が分割可能リソースで、それ以外の項が固定リソースです。

ログ容量の分割可能リソース = $(0.001[\text{MB}] \times 1 \text{日当たりのリクエスト数}) \times (\text{稼働情報保有日数} + 1) \times 2$ ログ容量の固定リソース = $8[\text{MB}] \times (\text{稼働情報保有日数} + 1) + 980\text{MB} \times 2$

ただし、最低でも1件のリクエストを処理するディスク容量は必要なので、それを最低必要リソースとします。

ログ容量の最低必要リソース = $(0.001[\text{MB}] \times 1 + 8[\text{MB}]) \times (\text{稼働情報保有日数} + 1) \times 2$

2.2 マシン台数の算出

2.2.1 マシンスペックの決定

マシン台数の算出に必要なマシンスペックを次に示します。

#		備考
1	CPU コア数	
2	搭載メモリーのうち Application Server が使用できる物理メモリーサイズ	最低でも 2GB は必要です。

2.2.2 マシン台数の算出

システム全体に必要なマシンリソースを確保できるようマシン台数を決定します。必要なマシンリソースを、マシンごとに分割できる分(分割可能リソース)と分割できない分(固定リソース)に分けると、次の式で必要なマシン台数が求められます。

$$\text{マシン台数[台]} = \frac{\text{分割可能リソース}}{\text{マシン1台が持つリソース} - \text{固定リソース}}$$

CPU、メモリーをどれだけ必要とするかは、業務によって異なります。そのため、次の3つの観点からそれぞれ必要なマシン台数を見積もり、大きいものを採用します。

- CPU コア数から見積もったマシン台数
- メモリーサイズ(New 領域基準)から見積もったマシン台数
- メモリーサイズ(Tenured 領域基準)から見積もったマシン台数

2.2.3 マシン 1 台当たりの Application Server が使用するリソース

決定したマシンスペックおよびマシン台数に従った、マシン 1 台当たりの Application Server が使用するリソースは、次の式で求められます。

$$\text{CPU コア数[個]} = \frac{\text{CPUコア数の分割可能リソース}}{\text{マシン台数}}$$

物理メモリーサイズ[MB]

$$\begin{aligned} &= (\text{JavaEE サーバが使用するメモリーサイズ}^{\ast 1} \\ &\quad + \text{Web サーバが使用するメモリーサイズ} \\ &\quad + \text{パフォーマンストレーサーが使用するメモリーサイズ} \\ &\quad) \times \text{ノード数} \\ &+ \text{ドメイン管理サーバが使用するメモリーサイズ} \end{aligned}$$

注※1 JavaEE サーバが使用するメモリーサイズ[MB]

$$\begin{aligned} &= \text{Java ヒープサイズ}^{\ast 2} \\ &\quad + \text{Explicit ヒープサイズ}^{\ast 3} \\ &\quad + \text{Metaspace サイズ} \\ &\quad + \text{C ヒープサイズ} + \text{スレッドスタックサイズ} \end{aligned}$$

注※2 Java ヒープサイズ[MB]

$$= \frac{\text{Java ヒープサイズの分割可能リソース}}{\text{ユニット数} \times \text{マシン台数}} + \text{Java ヒープサイズの分割不能リソース}$$

注※3 Explicit ヒープサイズ[MB]

$$= \frac{\text{Explicit ヒープサイズの分割可能リソース}}{\text{ユニット数} \times \text{マシン台数}} + \text{Explicit ヒープサイズの分割不能リソース}$$

注※2 および注※3 の Java ヒープサイズおよび Explicit ヒープサイズは、チューニングガイドでパラメーター値として使用します。

3 処理性能の求め方

この章では、処理性能の求め方について説明します。

本章の構成

- 3.1 処理性能の概算
- 3.2 処理性能の測定

3.1 処理性能の概算

アプリケーションで扱う情報量から、メモリーに関する処理性能を概算する方法を示します。CPUに関する処理性能は「3.2 処理性能の測定」を参考に測定します。

(1) 内部保留時間

概算見積もりでは1秒固定とします。

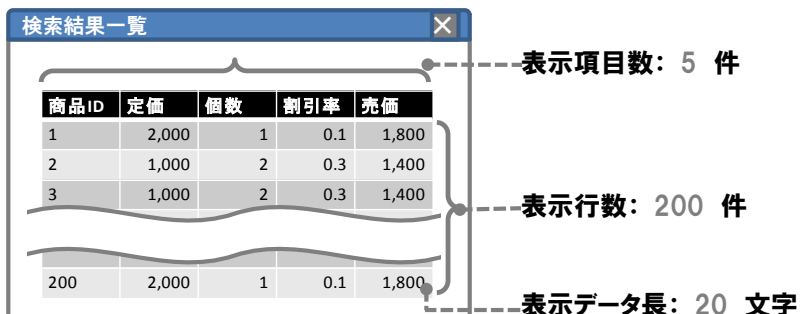
(2) 1スレッド使用 Survivor 領域サイズの概算

■算出に必要な情報

使用 Survivor 領域サイズの算出に必要な情報を次に示します。

#	カテゴリ	要素名	備考
1	参照業務で取得する情報量	表示項目数[件]	図 3.1-1 参照
2		表示行数[件]	図 3.1-1 参照
3		表示データ長[文字]	図 3.1-1 参照
4		XML データサイズ[MB]	図 3.1-1 参照

・表



・XML

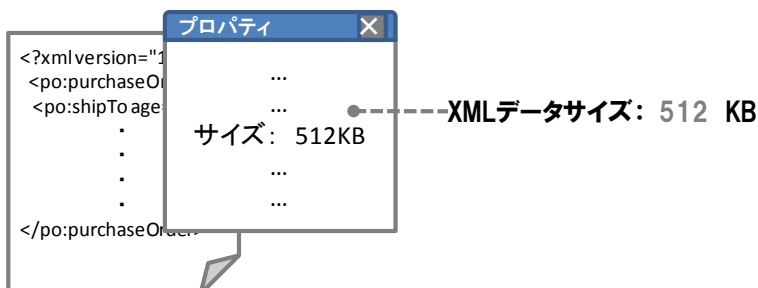


図 3.1-1 参照業務で取得する情報量

■算出方法

基礎値を使用し、次の式で使用 Survivor 領域サイズを算出できます。

$$1 \text{ スレッド使用 Survivor 領域サイズ[MB]} = \frac{(\text{表示データ長} \times 2 + 71) \times \text{表示項目数} \times \text{表示行数}}{1024 \times 1024} + \text{XML データサイズ[MB]} \times 20$$

(3) 共通使用 Survivor 領域サイズの概算

概算見積もりでは 1MB 固定とします。

(4) アプリケーション使用 Tenured 領域サイズの概算

概算見積もりでは 50MB 固定とします。

(5) 1セッション当たりの使用メモリーサイズの概算

■算出に必要な情報

1セッション当たりの使用メモリーサイズの算出に必要な情報を次に示します。

#	カテゴリ	要素名	備考
1	ユーザー情報量	文字数	図 3.1-2 参照
2		文字列数	図 3.1-2 参照

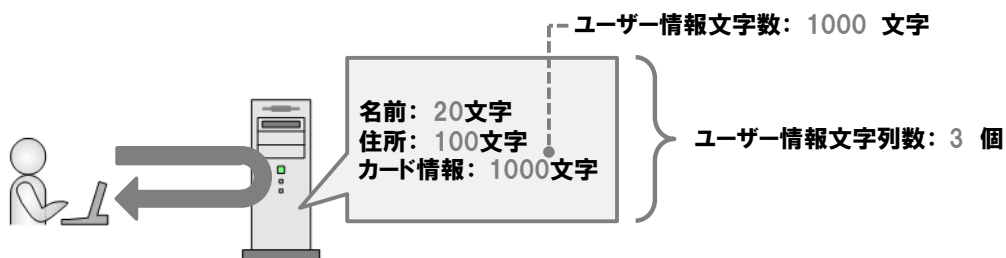


図 3.1-2 ユーザー情報量

■算出方法

基礎値を使用し、次の式で1セッション当たりの使用メモリーサイズを算出できます。

$$1 \text{ セッション当たりの使用メモリーサイズ[MB]} = \frac{(\text{文字数} \times 2 + 71 + 104) \times \text{文字列数} + 176}{1024 \times 1024}$$

3.2 処理性能の測定

3.2.1 実行 CPU 時間

(1) 測定方法

次の手順で測定します。

<測定手順の流れ>

- ① ドメイン管理サーバを停止する
- ② CPU 使用率を測定するコマンドを起動する
- ③ 測定対象のリクエストを投入する(業務処理を実行し、CPU を使用する)
- ④ CPU 使用率を測定するコマンドを終了する
- ⑤ ドメイン管理サーバを起動する

<測定手順の詳細>

- ① ドメイン管理サーバを停止する

次のコマンドを実行します。

コマンド	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin stop-domain</code>
------	---

- ② CPU 使用率を測定するコマンドを起動する

JavaEE サーバプロセス(java)の CPU 使用率を 1 秒間隔で取得するようにパフォーマンスモニターを設定します。

使用する OS によって、実行するコマンドが異なります。Linux の場合は次のコマンドを実行します。

コマンド	<code># top -d 1 -p `sbin/pidof java` -b > result※</code>
------	--

注※ *result* は任意のファイル名です。

AIX の場合は次のコマンドを実行します。

コマンド	<pre># while [true] > do > ps -e -o pid,pcpu,args grep /opt/hitachi/APServer/jdk/bin/java > result※ > sleep 1 > done</pre>
------	---

注※ *result* は任意のファイル名です。

- ③ 測定対象のリクエストを投入する(業務処理を実行し、CPU を使用する)

Web ブラウザーなどを使用します。投入したリクエスト数はカウントしておきます。

④ CPU 使用率を測定するコマンドを終了する

CTRL キーを押しながら C を押すなどして、CPU 使用率を測定するコマンドの実行を終了させます。

⑤ ドメイン管理サーバを起動する

次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin start-domain
------	---

(2) 算出方法

CPU 使用率を測定するコマンドの実行結果から、JavaEE サーバ[java]が使用した実行 CPU 時間[ms]を算出します。

CPU 使用率を測定するコマンドの実行結果には java の CPU 使用率[%CPU]が出力されています。1 秒間隔で取得した CPU 使用率であるため、100(%)で割ることで java が使用した実行 CPU 時間[s]になります。さらに 1000 を掛け、単位を msec にします。算出された実行 CPU 時間[ms]の和を投入したリクエスト数で割った値が、その業務の実行 CPU 時間[ms]になります。

$\text{実行 CPU 時間[ms]} = \text{java の CPU 使用率[\%CPU]} \div 100[\%] \times 1000 \div \text{リクエスト数}$

Linux での実行例を次に示します。

<top コマンドの実行結果>

(ヘッダーは省略)											
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3023	root	25	0	875m	135m	26m	S	0.0	13.4	3:06.34	java
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3023	root	25	0	875m	212m	26m	S	92.2	21.0	3:07.39	java
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3023	root	25	0	883m	231m	26m	S	99.0	22.9	3:08.39	java
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3023	root	25	0	875m	309m	26m	S	89.0	30.6	3:09.09	java
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3023	root	25	0	875m	309m	26m	S	0.0	30.6	3:09.09	java

<リクエスト数>

100 回

<実行 CPU 時間の算出>

$$(92.2 + 99.0 + 89.0) / 100 * 1000 / 100 = 28.02[\text{ms}]$$

3.2.2 内部保留時間

(1) 測定方法

次の手順で測定します。

<手順の流れ>

- ① 測定対象のリクエストを投入する
- ② アクセスログで内部保留時間を確認する

<手順の詳細>

① 測定対象のリクエストを投入する

測定対象のリクエストを実行します。リクエストの送信には **JMeter** などの負荷ツールを使用します。

② アクセスログで内部保留時間を確認する

アクセスログに内部保留時間が出力されているため、その最大値を確認します。

Application Server のインストールディレクトリーが `/opt/hitachi/APServer`、ノードの名称が `localhost-domain1`、Web サーバの名称が `Web1` の場合は、次のよう出力されます。

```
/opt/hitachi/APServer/javaee/logs/nodes/localhost-domain1/Web1/access.[10桁の数字*]
```

注※ [10桁の数字]が大きいものほど新しいファイルです。

アクセスログの例を次に示します。リクエストごとに1行のログが出力されます。

```
192.168.255.255 - - [24/Jul/2014:17:06:40.596 +0900] "GET /healthcheck/status HTTP/1.1" 200 103 0.046 6884 196.168.255.80/2664/0x0000000000000007
```

- 4列目：リクエスト処理を開始した時刻
- 5列目：HTTP通信のリクエストの先頭行
- 8列目：内部保留時間[秒]

上記の例の場合、リクエスト"GET /healthcheck/status HTTP/1.1"の内部保留時間は0.046秒となります。

3.2.3 1 スレッド使用 Survivor 領域サイズ、共通使用 Survivor 領域サイズ

(1) 測定方法

次の手順で測定します。

<手順の流れ>

- ① Java ヒープサイズを十分大きい値に設定する
- ② 2 つ以上の多重度で一連のリクエストを実行する
- ③ JavaVM ログファイルで使用 Survivor 領域サイズを確認する
- ④ 多重度と使用 Survivor 領域サイズを分析する

<手順の詳細>

① Java ヒープサイズを十分大きい値に設定する

JavaEE サーバへ 1GB を設定するために、Application Server のインストールディレクトリーが `/opt/hitachi/APServer`、JavaEE サーバの名称が `JavaEE1` の場合は、次のコマンドを実行します。

コマンド	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin stop-servers</code>
	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin list-jvm-options --target JavaEE1</code>
	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin delete-jvm-options --target JavaEE1 -Xmx 設定値 -Xms 設定値</code>
	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin create-jvm-options --target JavaEE1 -Xms1024m -Xmx1024m</code>
	<code>/opt/hitachi/APServer/javaee/glassfish/bin/asadmin start-servers</code>

② 2 つ以上の多重度で一連のリクエストを実行する

1,20,40 など、2 つ以上の多重度で一連のリクエストを実行します。リクエストの送信には JMeter などの負荷ツールを使用します。

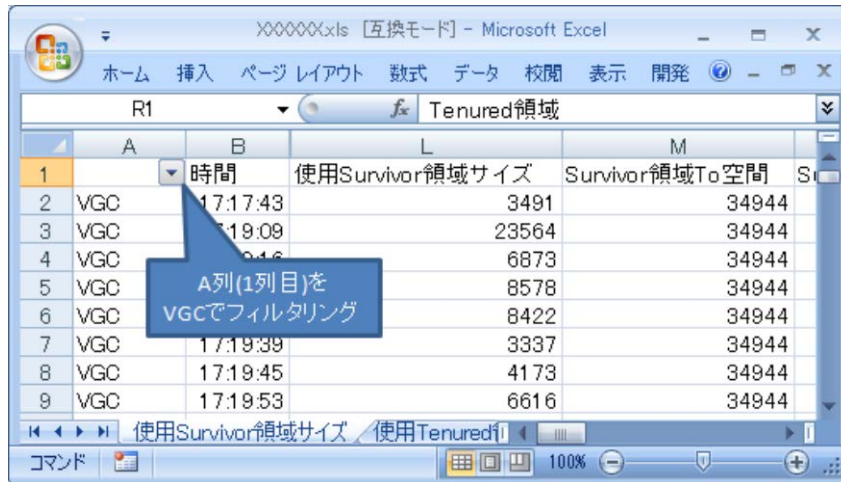
③ JavaVM ログファイルで使用 Survivor 領域サイズを確認する

JavaVM ログファイルに使用 Survivor 領域サイズが出力されているため、その最大値を確認します。ファイルの開始から数十行までは Application Server 自身のオブジェクトなどが含まれるため、50 行目以降を目安に値を確認します。また、使用 Survivor 領域サイズが Survivor 領域 To 空間の 1/2 を超えている場合、Survivor があふれているため Java ヒープサイズを大きくするか多重度を小さくして対応します。

Application Server のインストールディレクトリーが `/opt/hitachi/APServer`、ノードの名称が `localhost-domain1`、JavaEE サーバの名称が `JavaEE1` の場合は、次のように出力されます。

/opt/hitachi/APServer/javaee/glassfish/nodes/localhost-domain1/JavaEE1/javalog[nn].log

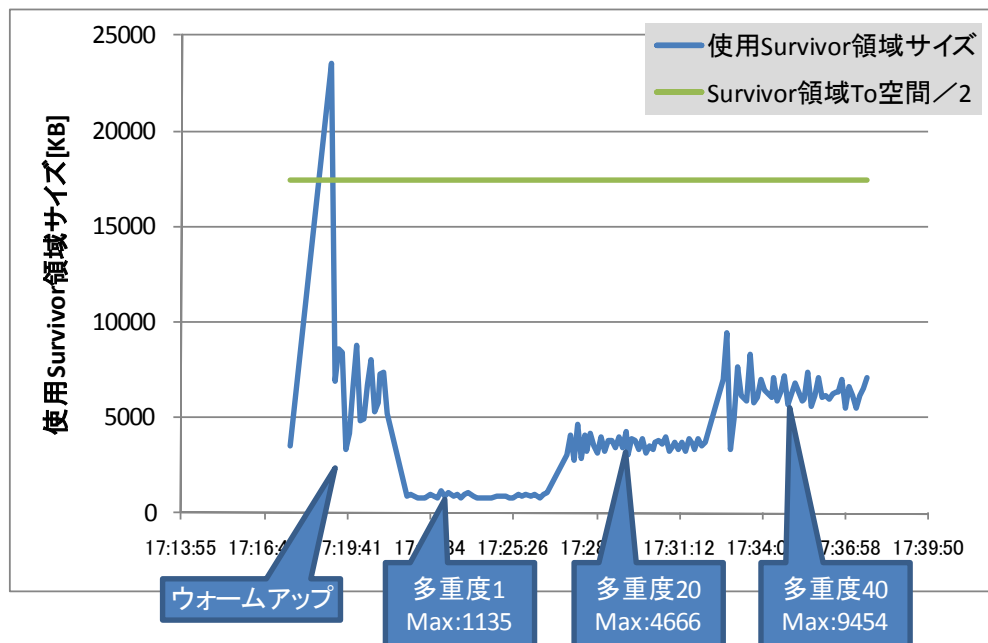
多重度 1, 20, 40 で測定した JavaVM ログファイルの例を次に示します。JavaVM ログファイルの拡張子を CSV とし Microsoft Office Excel で確認した例です。



	A	B	L	M
1		時間	使用Survivor領域サイズ	Survivor領域To空間
2	VGC	17:17:43	3491	34944
3	VGC	17:19:09	23564	34944
4	VGC	17:19:16	6873	34944
5	VGC	17:19:23	8578	34944
6	VGC	17:19:30	8422	34944
7	VGC	17:19:39	3337	34944
8	VGC	17:19:45	4173	34944
9	VGC	17:19:53	6616	34944

- A列(1列目) : JavaVM ログファイル識別子。VGC(GC の情報)を使用する。
- B列(2列目) : JavaVM ログを出力した時刻
- L列(12列目) : 使用 Survivor 領域サイズ[KB]
- M列(13列目) : Survivor 領域 To 空間サイズ[KB]

各多重度での、使用 Survivor 領域サイズの最大値を確認します。



④ 多重度と使用 Survivor 領域サイズを分析する

横軸を多重度、縦軸を使用 Survivor 領域サイズとした場合の傾きが、1 スレッド使用 Survivor 領域サイズです。切片が共通使用 Survivor 領域サイズです。

エクセルの数式を利用して、1 スレッド使用 Survivor 領域サイズ、共通使用 Survivor 領域サイズを算出した例を次に示します。1 スレッド使用の Survivor 領域サイズには SLOPE 関数、共通仕様 Survivor 領域サイズには INTERCEPT 関数を使用して算出しています。

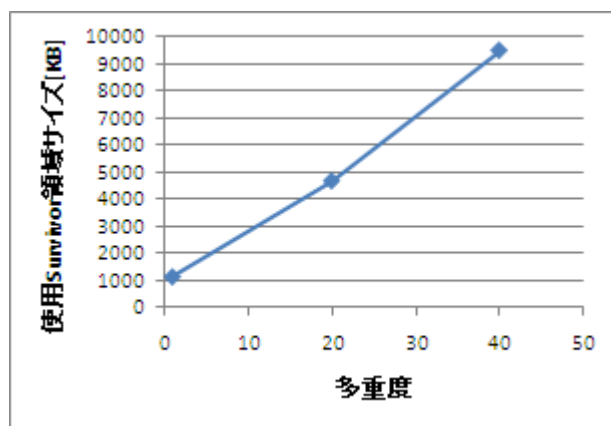
1スレッド使用Survivor領域サイズ, 共通使用Survivor領域サイズの測定

多重度と測定した使用Survivor領域サイズを入力します。

#	多重度	使用Survivor領域サイズ	単位
1	1	1135	KB
2	20	4666	KB
3	40	9454	KB

1スレッド使用Survivor領域サイズ, 共通使用Survivor領域サイズを算出します。

#	要素名	値	単位
1	1スレッド使用Survivor領域サイズ	0.209	MB
2	共通使用Survivor領域サイズ	0.726	MB



上記の例の場合、1 スレッド使用 Survivor 領域サイズ : 0.209MB、共通使用 Survivor 領域サイズ : 0.726MB となります。

3.2.4 アプリケーション使用 Tenured 領域サイズ

(1) 測定方法

次の手順で測定します。

<手順の流れ>

- ① Java ヒープサイズを十分大きい値に設定する
- ② アプリケーションに一連の操作を実行する
- ③ GC を発行する(短寿命オブジェクトを Tenured から削除する)
- ④ JavaVM ログファイルで使用 Tenured 領域サイズを確認する

<手順の詳細>

① Java ヒープサイズを十分大きい値に設定する

JavaEE サーバへ 1GB を設定するために、Application Server のインストールディレクトリーが/opt/hitachi/APServer、JavaEE サーバの名称が JavaEE1 の場合は、次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin stop-servers
	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin list-jvm-options --target JavaEE1
	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin delete-jvm-options --target JavaEE1 -Xmx 設定値-Xms 設定値
	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin create-jvm-options --target JavaEE1 -Xms1024m:-Xmx1024m
	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin start-servers

② アプリケーションに一連の操作を実行する

Web ブラウザーなどを使用します。

③ GC を発行する(短寿命オブジェクトを Tenured から削除する)

Application Server のインストールディレクトリーが/opt/hitachi/APServer の場合は、次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/jdk/jre/bin/javagc -p <JavaEE サーバのプロセス ID>
------	--

④ JavaVM ログファイルで使用 Tenured 領域サイズを確認する

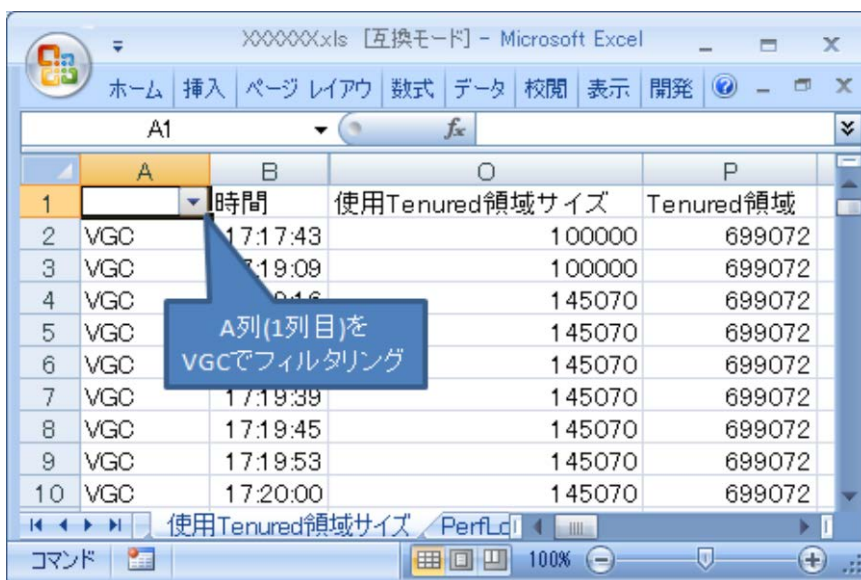
JavaVM ログファイルに使用 Tenured 領域サイズが出力されているため、③で発行した GC の情報を確認し、Application Server 使用 Tenured 領域サイズ(300MB)を引くことでアプリケーション使用 Tenured 領域サイズが確認できます。ただし、使用 Tenured 領域サイズが Application Server 使用 Tenured 領域サイズ(300MB)よりも小さい場合、

アプリケーション使用 Tenured 領域サイズは 0MB とします。

Application Server のインストールディレクトリーが /opt/hitachi/APServer、ノードの名称が localhost-domain1、JavaEE サーバの名称が JavaEE1 の場合は、次のように出力されます。

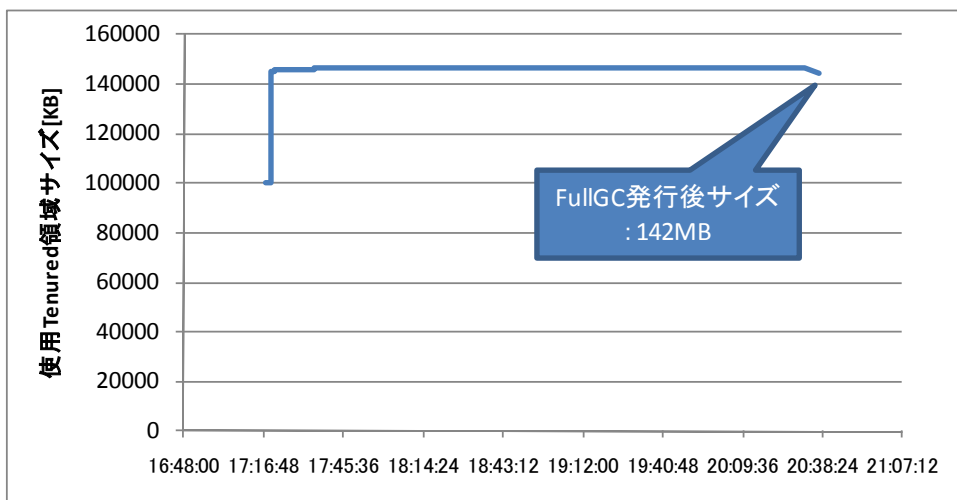
```
/opt/hitachi/APServer/javaee/glassfish/nodes/localhost-domain1/JavaEE1/javalog[nn].log
```

JavaVM ログファイルの例を次に示します。



- A 列(1 列目) : JavaVM ログファイル識別子。VGC(GC の情報)を使用する。
- B 列(2 列目) : JavaVM ログを出力した時刻
- O 列(15 列目) : 使用 Tenured 領域サイズ[KB]
- P 列(16 列目) : Tenured 領域サイズ[KB]

JavaVM ログファイルの Tenured 情報をグラフにした例を次に示します。



上記の例の場合、アプリケーション使用 Tenured 領域サイズは 42MB となります。

3.2.5 1セッション当たりの使用メモリーサイズ

(1) 測定方法

次の手順で測定します。

<手順の流れ>

- ① JavaEE サーバを再起動する(メモリーをリセットする)
- ② アプリケーションにログインし、ログアウトを除く一連の操作を実行する
- ③ GC を発行する(セッションオブジェクトを Explicit ヒープに移動させる)
- ④ アプリケーションを停止する(セッションオブジェクトを Explicit ヒープから解放する)
- ⑤ 稼働情報*を確認する

注※ Explicit ヒープから解放した、1セッション当たりの最大使用メモリーサイズを、60秒間隔でファイル出力している。

<手順の詳細>

- ① JavaEE サーバを再起動する(メモリーをリセットする)

Application Server のインストールディレクトリーが/opt/hitachi/APServer の場合は、次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin stop-servers
	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin start-servers

- ② アプリケーションにログインし、ログアウトを除く一連の操作を実行する
Web ブラウザーなどを使用します。

- ③ GC を発行する(セッションオブジェクトを Explicit ヒープに移動させる)

Application Server のインストールディレクトリーが/opt/hitachi/APServer の場合は、次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/jdk/jre/bin/javagc -p <JavaEE サーバのプロセス ID>
------	--

- ④ アプリケーションを停止する(セッションオブジェクトを Explicit ヒープから解放する)

Application Server のインストールディレクトリーが/opt/hitachi/APServer の場合は、次のコマンドを実行します。

コマンド	/opt/hitachi/APServer/javaee/glassfish/bin/asadmin disable --target JavaEE1 <アプリケーションの EAR 名>
------	--

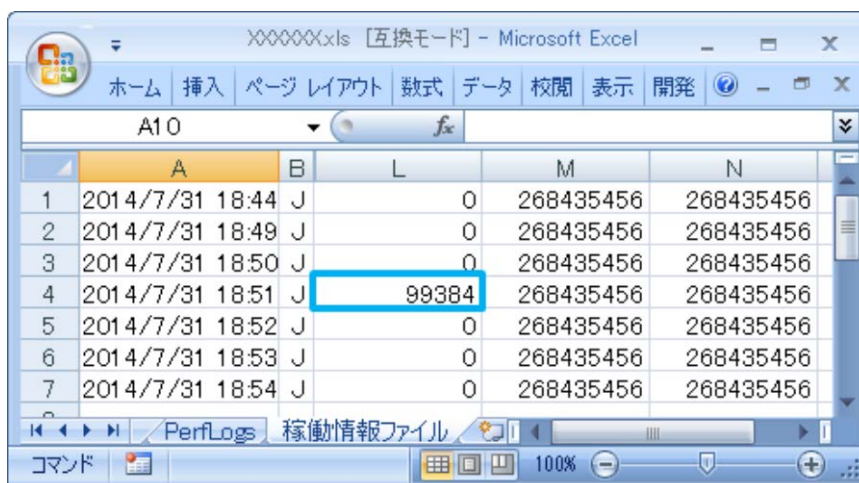
- ⑤ 稼働情報(Explicit ヒープから解放した、1セッション当たりの最大使用メモリーサイズを、60秒間隔でファイル出力している)を確認する

Java VM メモリー拡張稼働情報ファイルを確認します。Application Server のインストールディレクトリーが/opt/hitachi/APServer、ノードの名称が localhost-domain1、JavaEE サーバの名称が JavaEE1 の場合は、次のよう出力されます。

```
/opt/hitachi/APServer/javaee/logs/nodes/localhost-domain1/JavaEE1/statistics/JVMMemoryExtensionsStatisticsJavaEE1_<YYYYMMDDhhmm><TZ>.csv
```

Java VM メモリー拡張稼働情報ファイルに出力されている、「過去 60 秒間に Explicit ヒープから解放した 1 セッション当たりの最大使用メモリーサイズ[B]」を確認することで、1セッション当たりの使用メモリーを知ることができます。

稼働情報ファイルの例を次に示します。



	A	B	L	M	N
1	2014/7/31 18:44	J	0	268435456	268435456
2	2014/7/31 18:49	J	0	268435456	268435456
3	2014/7/31 18:50	J	0	268435456	268435456
4	2014/7/31 18:51	J	99384	268435456	268435456
5	2014/7/31 18:52	J	0	268435456	268435456
6	2014/7/31 18:53	J	0	268435456	268435456
7	2014/7/31 18:54	J	0	268435456	268435456

- A 列(1 列目) : 稼働情報を出力した時刻
- L 列(12 列目) : 過去 60 秒間に Explicit ヒープから解放した 1 セッション当たりの最大使用メモリーサイズ[B]
- アプリケーションを停止した時刻 : 18:50:40

上記の例の場合、1セッション当たりの使用メモリーは **97KB** となります。