HITACHI
Inspire the Next

Hitachi Application Server

チューニングガイド [Windows]

性能要件・信頼性要件を満たすためのパラメーター設計方法

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（Windows(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」の構築・運用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構築する立場にあるシステム構築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」のチューニングを実施できます。

２．対象とする製品

Hitachi Application Server 10-11

■製品名と機能名の表記

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

表記		製品名
Java EE Server		Hitachi Java EE Server
JavaVM ログファイル		日立 JavaVM ログファイル
Oracle	Oracle 11g	Oracle® Database 11g
	Oracle 11g R2	Oracle® Database 11g Release 2
	Oracle 12c	Oracle® Database 12c
Web Server		Hitachi Web Server
Windows	Windows Server 2008 R2	Microsoft® Windows Server® 2008 R2 Standard 日本語版
		Microsoft® Windows Server® 2008 R2 Enterprise 日本語版
		Microsoft® Windows Server® 2008 R2 Datacenter 日本語版
	Windows Server 2012	Microsoft® Windows Server® 2012 Standard 日本語版
		Microsoft® Windows Server® 2012 Datacenter 日本語版
	Windows Server 2012 R2	Microsoft® Windows Server® 2012 R2 Standard 日本語版
		Microsoft® Windows Server® 2012 R2 Datacenter 日本語版
拡張 DD		日立拡張 DD

■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	Java 関連用語
EJB	Enterprise JavaBeans™
Java	Java™
JavaEE	Java™ Platform, Enterprise Edition
JavaVM	Java™ Virtual Machine
JDBC	Java™ Database Connectivity
	JDBC™
JTA	JavaTM Transaction API

Servlet	Java™ Servlet

■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

チューニングガイド

性能要件・信頼性要件を満たすためのパラメーター設計方法

目　次

1 チューニング概要

この章では、Web フロントシステムのシステム構成とチューニングの流れを示します。

本章の構成

1.1 システム構成とチューニングの流れ

1.1 システム構成とチューニングの流れ

本書では、次の 2 点について説明します。

- 性能・信頼性確保のためのチューニングパラメーターの設計指針：「**2.　チューニングパラメーターの設計**」
  システムの性能・信頼性を確保するために設計が必要なパラメーターをチューニングパラメーターと呼びます。ここでは、システムの要件に沿ったチューニングパラメーターの設計指針を示します。
- システムがシステム要件を満たしているかの検証方法：「**3.　システム要件の検証**」
  サイジングが妥当であったかどうか、およびチューニングパラメーターが妥当であったかどうかを検証します。

システム構成とチューニングの流れを次に示します。

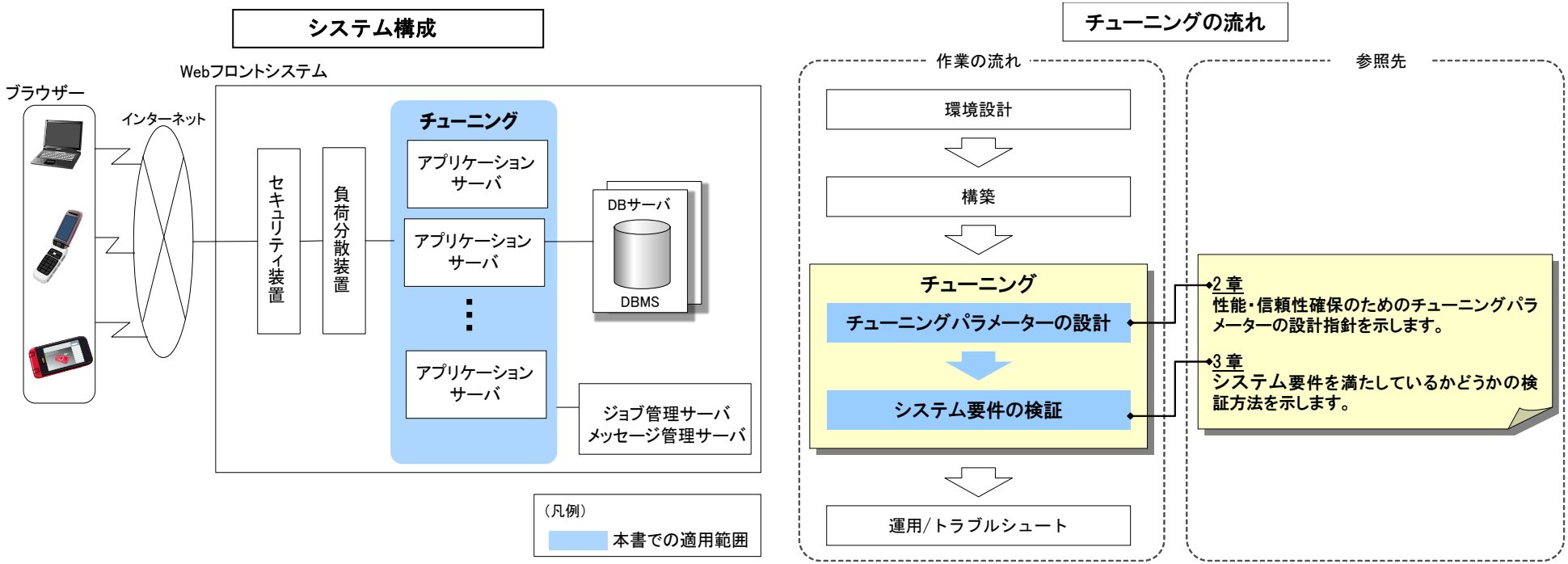

図 1.1-1　システム構成とチューニングの流れ

2 チューニングパラメーターの設計

この章では、チューニングパラメーターの設計指針およびパラメーター一覧について示します。

本章の構成

2.1 JavaVM メモリー

2.1.1 設計指針

●Java ヒープサイズ

Java ヒープは、Java EE サーバおよび Java EE サーバ上で動作するアプリケーションの各種オブジェクトを格納するメモリー領域です。Java ヒープサイズによって GC の発生間隔・GC に掛かる時間が変動し、性能・信頼性に影響します。

Java ヒープサイズには、サイジング時に決定した Java EE サーバ 1 台当たりの Java ヒープサイズを設定します。また、Java ヒープサイズには初期値と最大値に同じ値を設定し、Java EE サーバ起動時に必要なメモリー領域を確保するようにします。

●Explicit ヒープサイズ

Explicit ヒープは、Java EE サーバおよび Java EE サーバ上で動作するアプリケーションのセッション情報を格納するメモリー領域です。十分な Explicit ヒープサイズを確保することで、セッション情報が Java ヒープに格納されることに起因する FullGC の発生を防ぐことができます。

Explicit ヒープサイズには、サイジング時に決定した Java EE サーバ 1 台当たりの Explicit ヒープサイズを設定します。

●Metaspace サイズ

Metaspace は、Java EE サーバおよび Java EE サーバ上で動作するアプリケーションのクラスやメソッドなどのメタデータを格納する領域です。そのサイズは、およそ JavaVM 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。Metaspace が足りなくなると、OutOfMemoryError が発生し Java EE サーバがダウンします。なお，圧縮オブジェクトポインタ機能が有効な場合は，Metaspace 内に Compressed Class Space という領域が作成されます。

ロードされるクラスファイルの合計サイズから Metaspace サイズを見積もることはできますが、多くの場合、すべてのクラスがロードされることはなく、初期値で足ります。そのため、検証のフェーズですべての業務を実施し、足りなければ増やす方法を取ります。また、Metaspace サイズには初期値と最大値に同じ値を設定し、Java EE サーバ起動時に必要なメモリー領域を確保するようにします。また，JavaEE サーバを停止することなく Java EE サーバ上で動作するアプリケーションを入れ替える運用を行う場合には，Metaspace サイズに設定する値は，見積もり値に安全係数 3.0 を掛けた値を設定してください。これは，アンデプロイ処理後すぐに Metaspace に格納されたメタデータは解放されず， 2 回以上の FullGC を契機に削除されるためです。

Metaspace サイズが不足して FullGC が発生し、その後 OutOfMemoryError で Java EE サーバが異常終了した場合の拡張 GC ログの例を次に示します。

ログファイルパス（Application Server のインストールディレクトリーが C:¥Hitachi¥APServer、ノードの名称が localhost-domain1、Java EE サーバの名称が JavaEE1 の場合）：C:¥Hitachi¥APServer¥javaee¥glassfish¥nodes¥localhost-domain1¥JavaEE1¥config¥javalog[nn].log

```
[VGC]<Wed Jul 30 14:26:35.896 2014>[Full GC 99908K->81920K(1520448K), 0.7187500 secs][DefNew::Eden: 18270K->0K(419456K)][DefNew::Survivor: 0K->0K(52416K)][Tenured: 81637K->81920K(1048576K)][Perm: 262143K->262143K(262144K)][cause※1:MetaspaceAllocFail※2][User: 0.7187500 secs][Sys: 0.0000000 secs][IM: 61412K, 92064K, 0K][TC: 164][DOE: 1K, 5][CCI: 5872K, 49152K, 5952K]
[VGC]<Wed Jul 30 14:26:36.630 2014>[Full GC 81920K->81920K(1520448K), 0.7197266 secs][DefNew::Eden: 0K->0K(419456K)][DefNew::Survivor: 0K->0K(52416K)][Tenured: 81920K->81920K(1048576K)][Perm: 262143K->262143K(262144K)][cause※1:LastMetaspaceGC※3][User: 0.7031250 secs][Sys: 0.0000000 secs][IM: 61412K, 92064K, 0K][TC: 164][DOE: 1K, 5][CCI: 5872K, 49152K, 5952K]
[OOM][Thread: 0x000000000d051800]<Wed Jul 30 14:26:37.364 2014>[java.lang.OutOfMemoryError : requested 136 bytes. (Meta Space) : 164 threads exist]
```

注※1　VGC で始まる行の cause は、GC の要因を示しています。

注※2　MetaspaceAllocFail：Metaspace の割り付け失敗によって GC が発生したことを示します。

注※3　LastMetaspaceGC：Metaspace の OutOfMemory を出す前の最後の GC が発生したことを示します。

補足　Java ヒープサイズ設定での注意事項

ドメイン管理サーバは 1 時間ごとに、Java EE サーバは 24 時間ごとに SystemGC を実行します。SystemGC の実行間隔はシステムプロパティーの「sun.rmi.dgc.server.gcInterval」と「sun.rmi.dgc.client.gcInterval」で変更できます。これらのシステムプロパティーで SystemGC の発生間隔を広げても GC 発生回数が削減されない場合は、Java ヒープが不足している可能性があります。この場合は、Java ヒープのチューニングを実施することで改善することがあります。

2.1.2 パラメーター一覧

JavaVM メモリーのパラメーターは、asadmin の JavaVM オプション一覧表示コマンドで設定値を確認します。また、asadmin の JavaVM オプション削除コマンドと JavaVM オプション追加コマンドで設定値を変更できます。

パラメーター一覧を次に示します。

表 2.1-1 JavaVM メモリー パラメーター一覧

#	定義項目（初期値）		説明
	Windows(x86)	Windows(x64)	
1	-Xms**512m**	-Xms**1536m**	Java ヒープの初期サイズを設定します。512m は 512MB を表します。
2	-Xmx**512m**	-Xmx**1536m**	Java ヒープの最大サイズを設定します。512m は 512MB を表します。
3	-XX:HitachiExplicitHeapMaxSize=**64m**	-XX:HitachiExplicitHeapMaxSize=**512m**	Explicit ヒープの最大サイズを設定します。64m は 64MB を表します。
4	-XX:MetaspaceSize=**128m**	-XX:MetaspaceSize=**256m**	Metaspace の初期サイズを設定します。128m は 12MB を表します。
5	-XX:MaxMetaspaceSize=**128m**	-XX:MaxMetaspaceSize=**256m**	Metaspace の最大サイズを設定します。128m は 128MB を表します。
6	-XX:CompressedClassSpaceSize=	-XX:CompressedClassSpaceSize=	圧縮オブジェクトポインタ機能が有効な場合に Metaspace 内に作られる領域のサイズを設定します。-XX:MaxMetaspaceSize と同じ値を設定します。

-XX:MaxMetaspace は Metaspace の最大サイズを設定するパラメータですが，設定したサイズを必ず使用するわけではありません。そのため，見積もり時に想定していなかった Metaspace の使用量の増加を考慮し，見積もり値より大きい値を設定することで，Metaspace の OutOfMemoryError が発生するリスクを下げることができます。ただし，設定した値までマシンのリソースを使用する場合があるため，他のプロセスへの影響を考慮して値を設定して下さい。

2.2 セッション

2.2.1 設計指針

●セッション最大数

セッション情報は Java EE サーバの Explicit ヒープ領域に格納されますが、Explicit ヒープ領域を使い切った場合、Java ヒープ領域に格納されます。そのため、セッションが上限なく生成されると、セッション情報が Java ヒープ領域を圧迫し、FullGC が発生するため性能・信頼性に影響します。

セッション最大数は、Java EE サーバで生成されるセッションの上限を Web アプリケーション（WAR）単位に設定するパラメーターです。セッションを利用する Web アプリケーション(WAR)がある場合、その Web アプリケーション（WAR）のセッション最大数に、サイジング時に決定した「同時ログイン数÷Java EE サーバ数」を設定します。

セッションが最大数に達した状態で、利用者からログイン要求が来た場合の動作は次のとおりです。

利用者への応答	Java EE サーバが HTTP ステータスコード 500(Internal Server Error)を返します。
ログ	Java EE サーバのメッセージログに次のメッセージを出力します。 StandardWrapperValve[<サーブレット名>]: Servlet.service() for servlet <サーブレット名> threw exception java.lang.IllegalStateException: createSession: Too many active sessions

2.2.2 パラメーター一覧

セッションのパラメーターは、asadmin の get サブコマンドで設定値を確認します。また、asadmin の set サブコマンドで設定値を変更できます。
パラメーター一覧を次に示します(Java EE サーバの名称が JavaEE1 の場合)。
拡張 DD を使用すると Java EE アプリケーションや Java EE モジュールごとに、パラメーターの設定を行うことができます。

表 2.2-1 セッション パラメーター一覧

#	定義項目（初期値：″=xx″または″（xx）″）	説明
1	configs.config.*JavaEE1*-config.web-container.session-config.session-manager.manager-properties.max-sessions=-1	セッションの最大数。-1～2147483647 の間で指定します。-1 を指定すると無制限となります。
2	拡張 DD(WAR): hitachi-web.xml /hitachi-web/session-config/session-manager/manager-properties/property/@name @name="maxSessions" (-1)	

2.3 流量制御

2.3.1 設計目的

システムは要件どおりリクエストを処理できるようにサイジングしています。しかし、実際の運用時には想定外のリクエストが来ることもあり、マシンの処理能力を超えると、安定したサービスを提供できなくなります。

流量制御パラメーターは、システムで処理するリクエストの数を制御するパラメーターです。このパラメーターを設定して、想定外の負荷状態でもシステムが安定稼働するようにします。

流量制御パラメーターを変更した場合、データベースの設定にも変更が必要です。

2.3.2 設計パラメーター

処理の流れと流量制御パラメーターの一覧を次に示します。図中の①～⑥は設計する順番です。

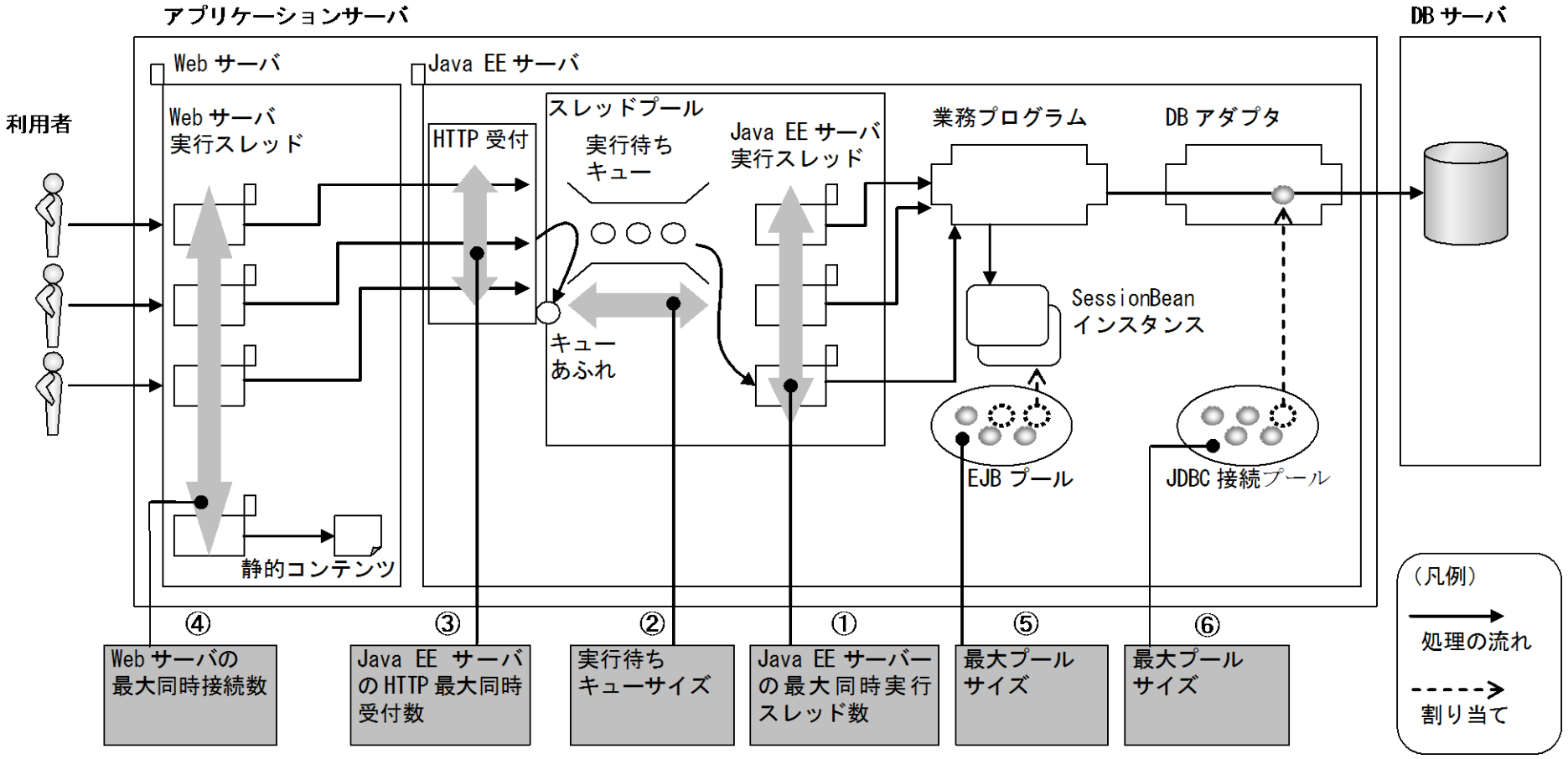

図 2.3-1 処理の流れと流量制御パラメーターの一覧

Java EE サーバの最大同時実行スレッド数（①）を設定することで、処理するリクエストを要件どおりに制限します。それを超える想定外のリクエストは、ある程度は実行待ちキュー（②）にいったん格納することで対処しますが、キューもあふれるような場合は、HTTP の場合には HTTP ステータスコード 503 (Service Unavailable) を返しシステムが混雑していることを利用者に通知します。その他のパラメーター（③～⑥）は、Java EE サーバの最大同時実行スレッド数（①）、実行待ちキューサイズ（②）分のリクエストが滞りなく処理されるように設定します。

2.3.3 設計指針

●Java EE サーバの最大同時実行スレッド数

Java EE サーバは、Web サーバから転送されたリクエストを受け取ると、1 リクエストに対し 1 スレッドを割り当てて処理を実行します。マシンの処理性能を超えたスレッド数で処理を実行すると、動作が不安定になります。

Java EE サーバの最大同時実行スレッド数は、Java EE サーバで同時に動作するスレッド数の上限を設定するパラメーターです。このパラメーターに、サイジング時に想定した処理性能を設定することで、想定外の負荷状態でもシステムが安定稼働するようにします。

サイジング時に想定した処理性能から Java EE サーバの最大同時実行スレッド数を算出する式を次に示します。なお，WebSocket を使用する場合、HTTP リクエスト数に加えて WebSocket フレーム受信数も考慮する必要があります。

Java EE サーバの最大同時実行スレッド数 ＝Java EE サーバの 1 秒当たりのリクエスト数[件/秒]※1×内部保留時間[秒] ※2 + Java EE サーバの 1 秒当たりの WebSocket フレーム受信数[件/秒]×内部保留時間[秒]・・・・ 【WebSocket を使用する場合のみ】 注※1 Java EE サーバの 1 秒当たりのリクエスト数[件/秒]≡1 秒当たりのリクエスト数[件/秒]/Java EE サーバ数 ※2 Java EE サーバの実行スレッドの実行時間

上記の式の考え方を次に示します。

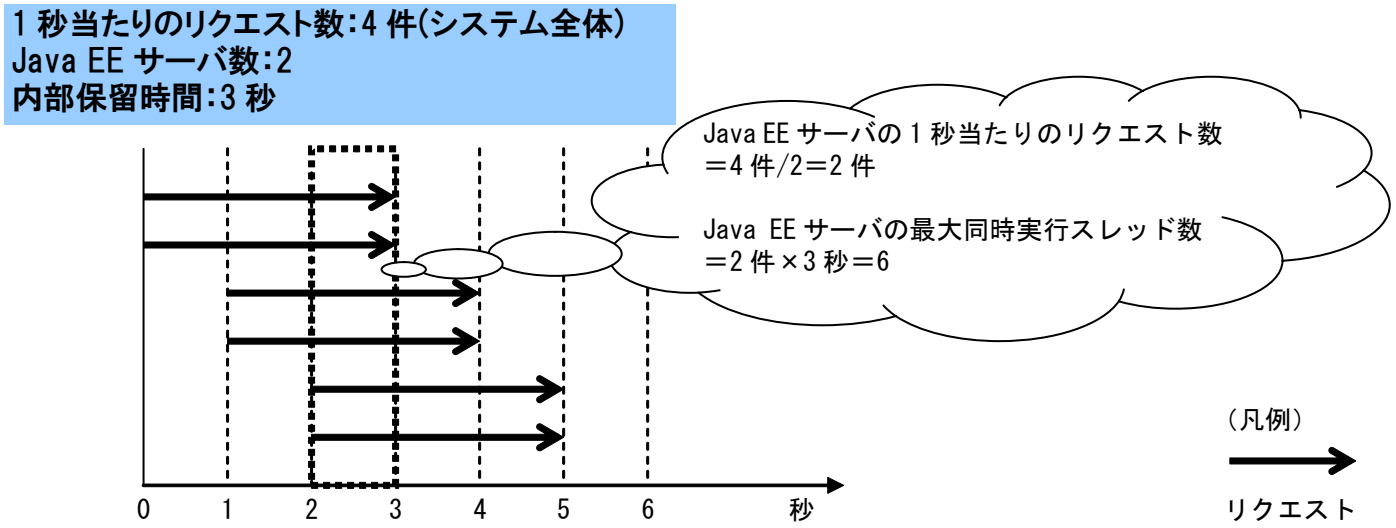

図 2.3-2 Java EE サーバの最大同時実行スレッド数の考え方

●Java EE サーバの最小実行スレッド数

指定された実行数分のスレッドが Java EE サーバ起動時に生成されます。起動時に必要となるリソースを取得し安定した性能を確保するため、Java EE サーバの最大同時実行スレッド数と同数とるすことをお勧めします。

●実行待ちキューサイズ

受け付けたリクエストは一旦実行待ちキュー内に入り、実行スレッドに空きがあれば実行スレッドに割り当てられます。

Java EE サーバの同時実行スレッド数が上限に達している場合、リクエストは実行待ちキュー内で実行スレッドが空くのを待ちます。

要件どおりのリクエストであれば実行待ちキューに滞留することなく処理できるようにサイジングしますが、実際の運用時には突発的なリクエストや、想定外のリクエストが来ることもあります。それらの場合に対応するため、実行待ちキューを使用します。

- 突発的なリクエストが来た場合⇒リクエストを一時的に格納して正常に処理する

  本書では、Java EE サーバの 1 秒当たりのリクエスト数から Java EE サーバの最大同時実行スレッド数を算出します。しかし、突発的に、リクエスト数の到着間隔が狭まり Java EE サーバの 1 秒当たりのリクエスト数が増加した場合、Java EE サーバの実行スレッド数が不足し、リクエストはエラーとなります。この場合、実行待ちキューを用意することで、レスポンス時間への影響を抑えて、エラーを回避できます。

上記の解説を次に示します。

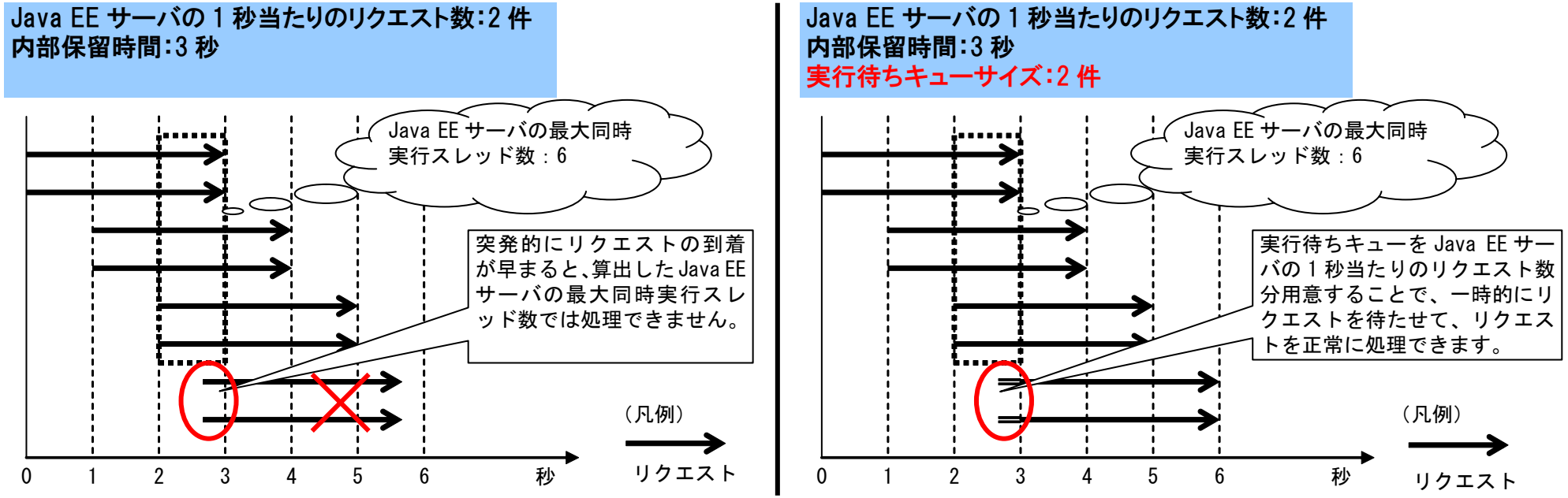

図 2.3-3 実行待ちキューサイズの解説 1

- 想定外のリクエストが来た場合⇒レスポンスを遅らせて正常に処理する、HTTP リクエストの場合は HTTP ステータスコード 503 (Service Unavailable)を返しシステムが混雑していることを通知する

  想定外のリクエストを実行待ちキューに入れることで、レスポンスは遅くなりますが、エラーにしないで処理できます。また、キューからあふれた HTTP リクエストに対しては、HTTP ステータスコード 503 (Service Unavailable)を返しシステムが混雑していることを通知できます。

上記の解説を次に示します。

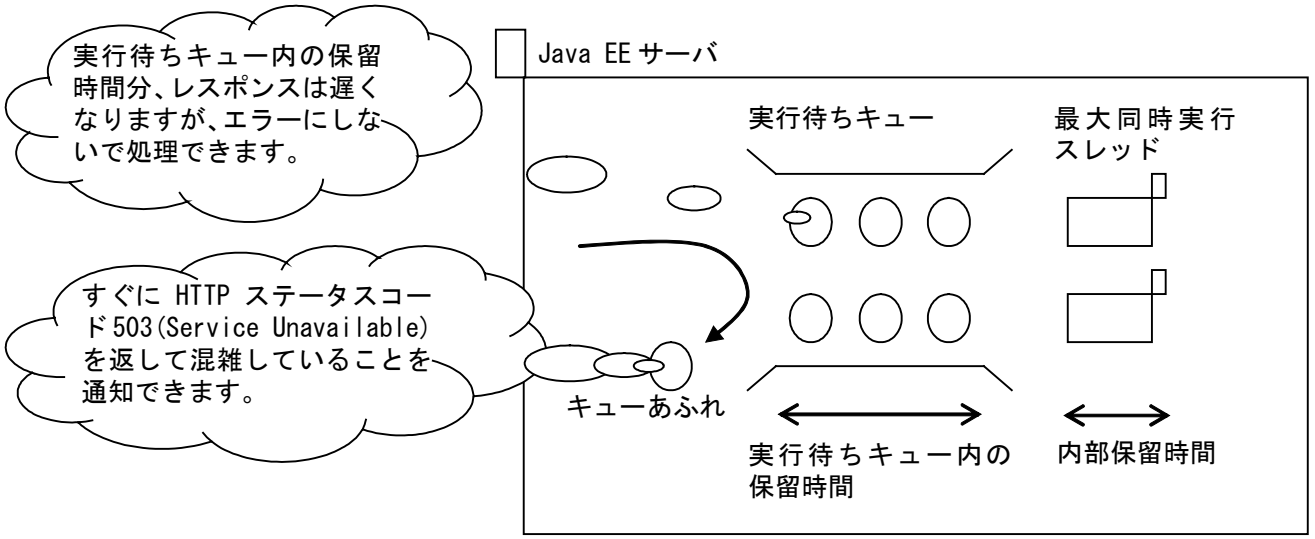

図 2.3-4 実行待ちキューサイズの解説 2

実行待ちキューサイズを算出する式を次に示します。

突発的なリクエストを一時的に格納するために、最低でも Java EE サーバの最大同時実行スレッド数分は用意します。

実行待ちキューサイズ≧Java EE サーバの最大同時実行スレッド数

それ以上は、想定外のリクエストに対し、どのくらいの実行待ちキュー内の保留時間を許容して正常に処理を行うかがチューニングのポイントになります。

実行待ちキュー内の保留時間＝実行待ちキューサイズ÷Java EE サーバの 1 秒当たりの HTTP リクエスト数 よって、 実行待ちキューサイズ＝実行待ちキュー内の保留時間×Java EE サーバの 1 秒当たりの HTTP リクエスト数

WebSocket を使用する場合は、以下の式となります。

実行待ちキュー内の保留時間＝ 実行待ちキューサイズ÷(Java EE サーバの 1 秒当たりの HTTP リクエスト数+ Java EE サーバの 1 秒当たりの WebSocket フレーム受信数) よって、 実行待ちキューサイズ＝ 実行待ちキュー内の保留時間×(Java EE サーバの 1 秒当たりの HTTP リクエスト数+ Java EE サーバの 1 秒当たりの WebSocket フレーム受信数)

上記の式の考え方を次に示します。

実行待ちキュー内の保留時間:6 秒
Java EE サーバの 1 秒当たりのリクエスト数:2 件
⇒実行待ちキューサイズ:6×2=12 件

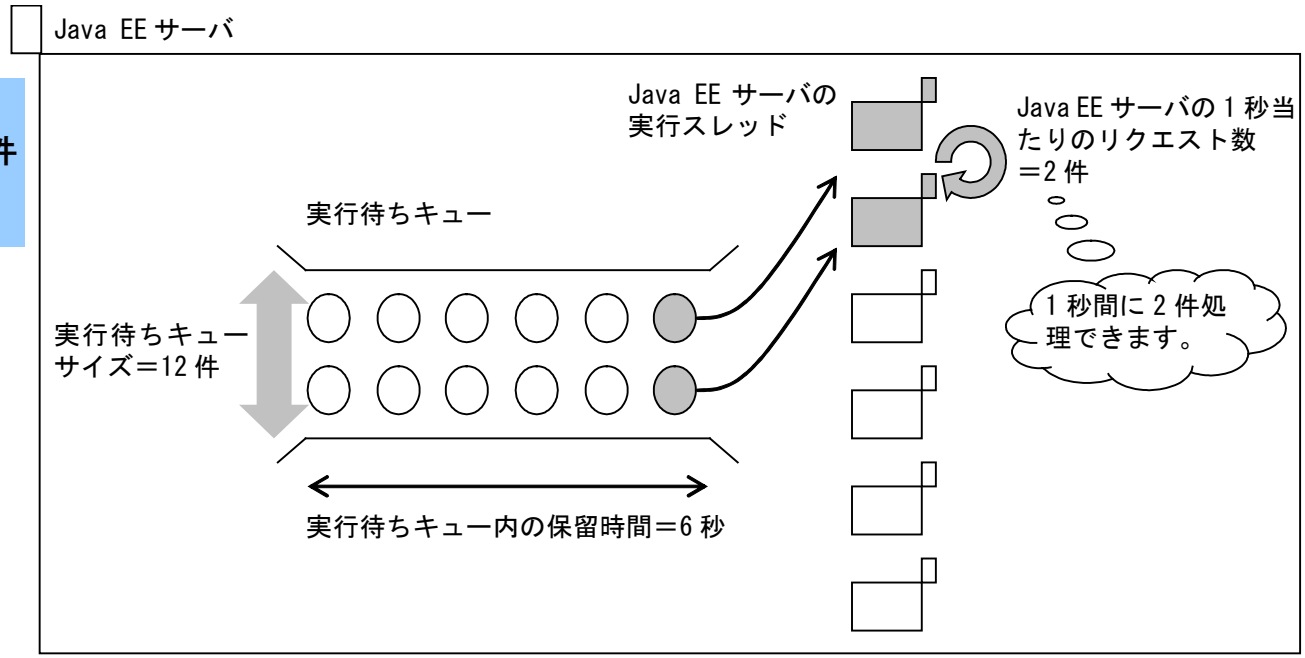

図 2.3-5 実行待ちキューサイズの考え方

●Java EE サーバの HTTP 最大同時受付数

Java EE サーバは、Web サーバから転送されたリクエストを受け取ると、リクエストに対する処理を行うスレッドを割り当てて処理を実行します。また、Java EE サーバの最大同時実行スレッド数が上限に達している場合、実行待ちキューに登録します。そのため、Java EE サーバの HTTP 最大同時受付数は、実行スレッドの上限値と実行待ちキューサイズの 2 つを考慮する必要があります。

Java EE サーバの HTTP 最大同時受付数を算出する式を次に示します。

Java EE サーバの HTTP 最大同時受付数 = Java EE サーバの最大同時実行スレッド数 + 実行待ちキューサイズ

●Web サーバの最大同時接続数

Web サーバは利用者からリクエストを受け取ると、1 リクエストに対し 1 スレッドを割り当てて処理を実行します。

Web サーバの最大同時接続数は、Web サーバで同時に動作するスレッド数の上限を設定するパラメーターです。このパラメーターには、Java EE サーバの最大同時実行スレッド数、および実行待ちキューサイズ分のリクエストが、滞りなく Java EE サーバに到達するために必要な数を設定します。これにより、想定外のリクエストに対して、実行待ちキューを使用した制御ができます。

Web サーバの最大同時接続数を算出する式を次に示します。

Web サーバの最大同時接続数
＝Java EE サーバの最大同時実行スレッド数
　＋Java EE サーバの 1 秒当たりのリクエスト数×3 秒　　…HTTP Keep Alive(3 秒)によって占有されるスレッド数。静的コンテンツに対する要求の応答用。
　＋実行待ちキューサイズ
　＋1　　…キューあふれによる、HTTP ステータスコード 503 (Service Unavailable)応答用スレッド

なお、WebSocket を使用する場合は、前提ソフトウェアである Microsoft(R) Internet Information Services の見積もり方法に従って算出してください。

上記の式の考え方を次に示します。

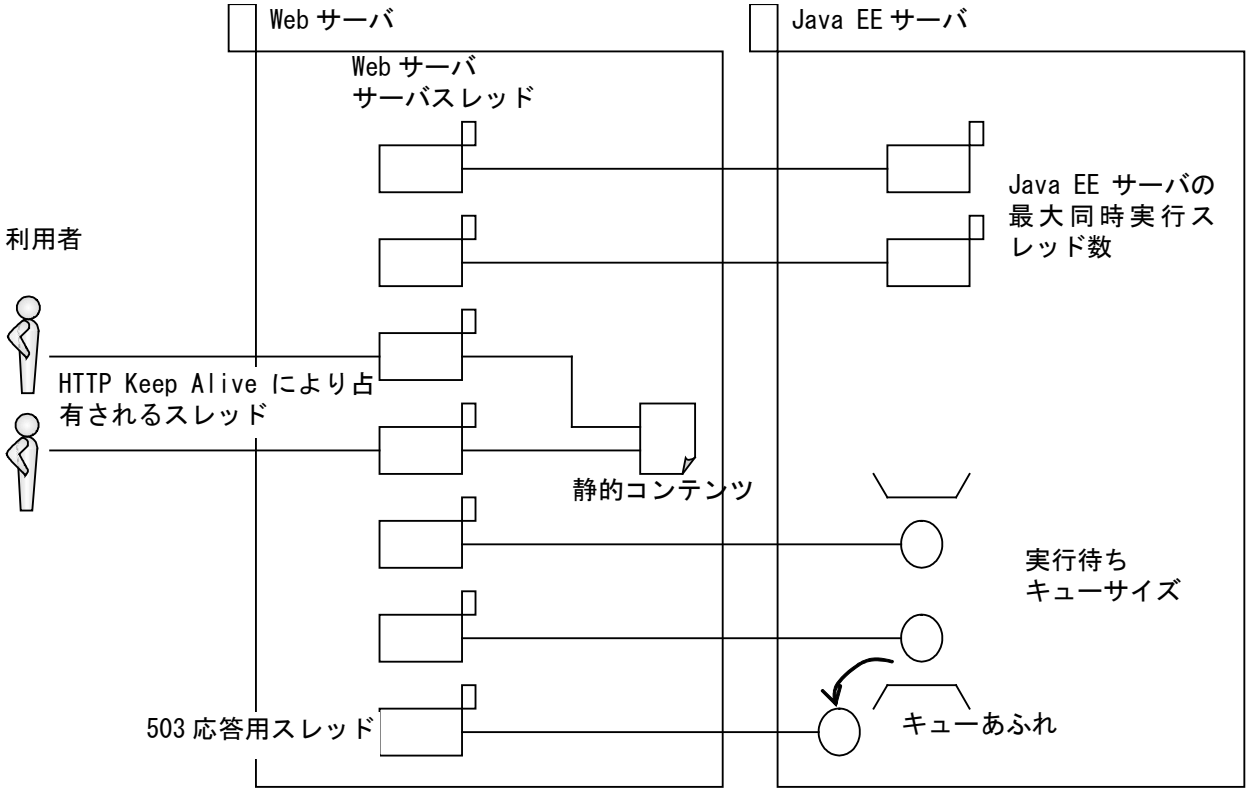

図 2.3-6 Web サーバの最大同時接続数の考え方

●EJB プールサイズ

Java EE サーバは SessionBean をプールすることで、性能の向上を図ります。通常、Java EE サーバの実行スレッド 1 つ当たりに、1 つの SessionBean を使用するため、Java EE サーバの最大同時実行スレッド数と同数の Bean をプールすれば、すべてのリクエストがプールから Bean を取得できることになります。

EJB プールサイズは、SessionBean をプールする数を設定するパラメーターで、個々の SessionBean に対し、最大値と最小値を設定できます。Java EE サーバ起動時に最小値分の Bean が用意され、最大値まで増加します。起動時に必要となるリソースを取得し安定した性能を確保するため、すべての SessionBean の最大値と最小値に対し、Java EE サーバの最大同時実行スレッド数を設定します。

EJB プールサイズを算出する式を次に示します。

EJB プール最大サイズ＝EJB プール最小サイズ＝Java EE サーバの最大同時実行スレッド数

●JDBC コネクションプールサイズ

Java EE サーバは JDBC コネクションをプールすることで、性能の向上を図ります。Java EE サーバの実行スレッド 1 つ当たりに、1 つのコネクションを使用するため、Java EE サーバの最大同時実行スレッド数と同数の JDBC コネクションをプールすれば、すべてのリクエストがプールから JDBC コネクションを取得できることになります。

JDBC コネクションプールサイズは、JDBC コネクションをプールする数を設定するパラメーターで、最大値と最小値を設定できます。Java EE サーバ起動時に最小値分の JDBC コネクションが用意され、最大値まで増加します。起動時に必要となるリソースを取得し安定した性能を確保するため、最大値と最小値に対し、Java EE サーバの最大同時実行スレッド数を設定します。

JDBC コネクションプールサイズを算出する式を次に示します。

JDBC コネクションプール最大サイズ＝JDBC コネクションプール最小サイズ＝Java EE サーバの最大同時実行スレッド数

2.3.4 パラメーター一覧

流量制御のパラメーターは、asadmin の get サブコマンドで設定値を確認します。また、asadmin の set サブコマンドで設定値を変更できます。
パラメーター一覧を次に示します（Web サーバの名称が WebServer1、Java EE サーバの名称が JavaEE1 の場合）。
拡張 DD を使用すると Java EE アプリケーションや Java EE モジュールごとに、パラメーターの設定を行うことができます。

表 2.3-1 流量制御 パラメーター一覧

#	定義項目（初期値：″=xx″または″（xx）″）	説明
1	hitachi-webservers.hitachi-webserver.*WebServer1*.property.threads-per-child=100	Web サーバの最大同時接続数。
2	configs.config.*JavaEE1*-config.network-config.protocols.protocol.http-listener-1.http.max-connections=256	Java EE サーバの HTTP 最大同時受付数。1～2147483647 の間で指定します。
3	configs.config.*JavaEE1*-config.thread-pools.thread-pool.http-thread-pool.max-thread-pool-size=24	Java EE サーバの最大同時実行スレッド数。1～2147483647 の間で指定します。指定された実行数分までスレッドが生成されます。
4	configs.config.*JavaEE1*-config.thread-pools.thread-pool.http-thread-pool.min-thread-pool-size=24	Java EE サーバの最小実行スレッド数。1～2147483647 の間で指定します。指定された実行数分のスレッドが Java EE サーバ起動時に生成されます。
5	configs.config.*JavaEE1*-config.thread-pools.thread-pool.http-thread-pool.max-queue-size=80	実行待ちキューサイズ。0～2147483647 の間で指定します。
6	configs.config.*JavaEE1*-config.ejb-container.max-pool-size=64	EJB プールの最大値。0～2147483647 の間で指定します。SessionBean ごとに設定します。
7	拡張 DD(EJB-JAR):hitachi-ejb-jar.xml /hitachi-ejb-jar/enterprise-beans/ejb/bean-pool/max-pool-size （64）	
8	configs.config.*JavaEE1*-config.ejb-container.steady-pool-size=32	EJB プールの最小値。0～2147483647 の間で指定します。SessionBean ごとに設定します。
9	拡張 DD(EJB-JAR):hitachi-ejb-jar.xml /hitachi-ejb-jar/enterprise-beans/ejb/bean-pool/steady-pool-size（32）	

また、コネクションプールのパラメーターは、asadmin の create-jdbc-connection-pool サブコマンドでコネクションプール作成時に設定してください。設定したパラメーターは、asadmin の get サブコマンドで設定値を確認できます。また、asadmin の set サブコマンドで設定値を変更できます。

パラメーター一覧を次に示します（コネクションプールの名称が ConnectionPool1 の場合）。

表 2.3-2 コネクションプールのパラメーター一覧

#	定義項目（初期値：″=xx″または″（xx）″）	説明
1	asadmin create-jdbc-connection-pool サブコマンド: --steadypoolsize オプション （24）	データベースサーバ接続用コネクションプール最小サイズ。0～2147483647 の間で指定します。
2	resources.jdbc-connection-pool.*ConnectionPool1*.steady-pool-size=24	
3	asadmin create-jdbc-connection-pool サブコマンド: --maxpoolsize オプション（24）	データベースサーバ接続用コネクションプール最大サイズ。1～2147483647 の間で指定します。
4	resources.jdbc-connection-pool.*ConnectionPool1*.max-pool-size=24	

2.4 タイムアウト（Oracle）（SQL Server）

2.4.1 設計目的

Java EE サーバおよびデータベースサーバで、処理遅延、通信遅延の障害が発生した場合、次のような問題が発生します。

- 業務処理が滞留し、その業務処理がスレッドやテーブル/レコードなどのリソースを占有し、ほかの業務処理に影響を与える
- 利用者を不要に待たせる

このような問題が発生する場合、システムでは次の対処を実施する必要があります。

- 遅延を検知
- 業務処理が占有しているリソースを解放
- 利用者へのエラー応答

タイムアウトパラメーターは、遅延を検知し、リソースの解放、利用者へのエラー応答をするためのパラメーターです。このパラメーターを設定して、処理遅延や通信遅延の障害が発生した場合でも、システムや利用者への影響を軽減するようにします。

2.4.2 設計パラメーター

タイムアウトパラメーターは 3 つあります。遅延の検知を複数の個所で実施することで、遅延が発生した個所の絞り込みが容易になります。各タイムアウトパラメーターの説明と、それぞれのタイムアウト時の動作を次に示します。

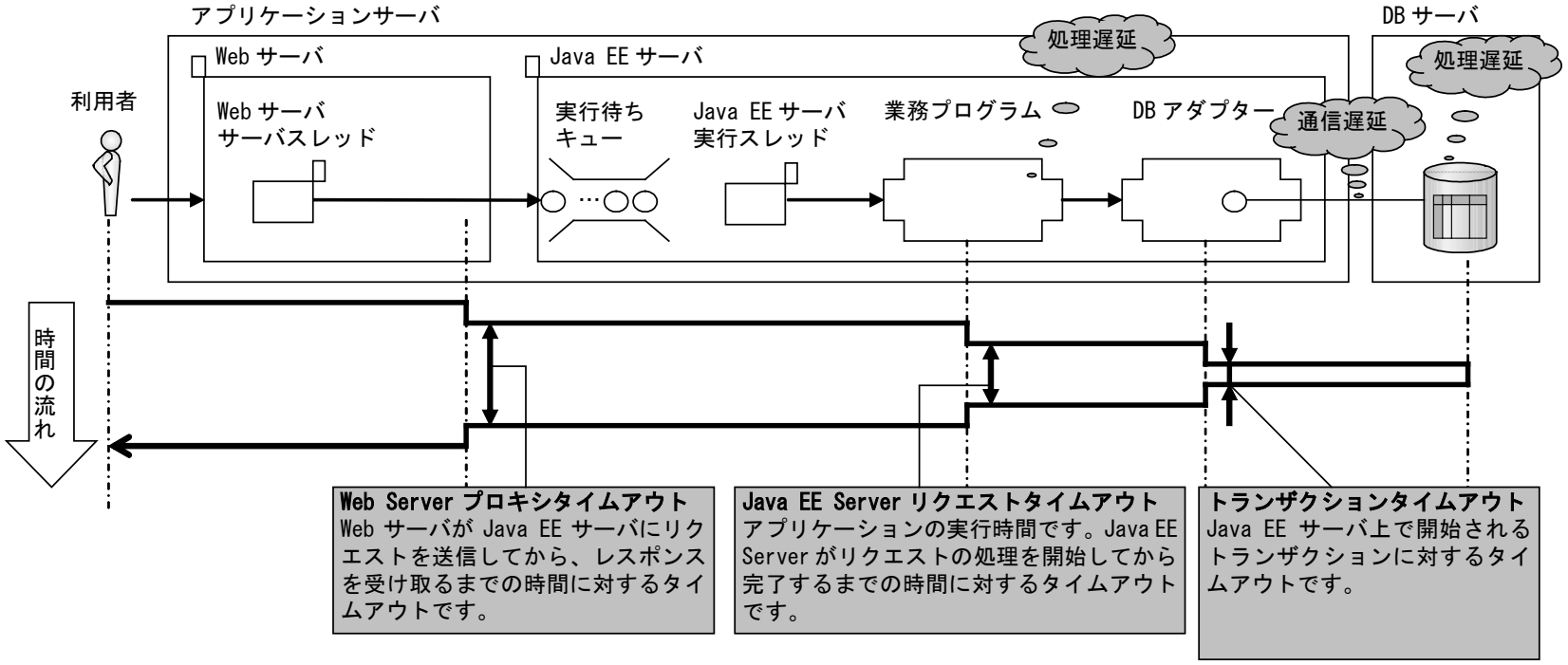

図 2.4-1 タイムアウトパラメーター

表 2.4-1 タイムアウト時の動作

#	パラメーター	利用者へのエラー応答	リソース解放	ログ
1	Web Server プロキシータイムアウト	HTTP ステータスコード 502(Proxy Error)を返します。	Web サーバのサーバスレッド、およびコネクションを解放します。(ただし、Java EE サーバ内の処理は継続します。)	Web サーバのログにメッセージ(KH00363 または KH00364)を出力。ただし、メッセージに含まれる接続先が JavaEE サーバである場合です。
2	Java EE Server リクエストタイムアウト	なし(アプリケーションの実行スレッドに対して割り込みが発生します。具体的には、Java EE Server が実行スレッドに対して、Thread#interrupt()メソッドを呼び出します。そのため，Java EE Server リクエストタイムアウトが発生するとアプリケーションで例外が発生することがあります。interrupt()メソッドについては、javadoc を参照してください。)。	Java EE サーバのスレッド、JDBC コネクションおよびデータベースサーバのテーブル/レコードを解放します。	ー
3	トランザクションタイムアウト	なし（JTA インタフェースを利用してトランザクション制御している場合はアプリケーションに例外が返ります)。	ー	ー

（凡例） ー：該当なし

2.4.3 設計指針

<u>利用者へのエラー応答を目的としたパラメーター</u>

●Web Server プロキシータイムアウト

Web サーバ以降（JavaEE サーバ、データベースサーバ）の処理で、遅延障害が発生していると判断する時間を設定します。

Web Server プロキシータイムアウト＝Web サーバ以降の処理で遅延障害が発生していると判断する時間

また、実行待ちキューサイズの設計時に考慮した、「許容できる内部保留時間」よりも大きい値を設定してください。小さい値を設定すると、キューに格納したリクエストの処理終了を待たないで、Web サーバでタイムアウトする場合があります。

Web Server プロキシータイムアウト＞許容できる内部保留時間＝内部保留時間＋実行待ちキュー内の保留時間

●トランザクションタイムアウト

全トランザクションの処理時間を踏まえ、明らかにトランザクション処理中に異常な遅延が発生していると判断する時間を設定します。

トランザクションタイムアウト＝トランザクション処理中に異常な遅延が発生していると判断する時間

リソースの解放を目的としたパラメーター

●Java EE Server リクエストタイムアウト

全業務の処理時間を踏まえ、明らかに業務内で異常な遅延が発生していると判断する時間を設定します。これによって、Java EE サーバのスレッド、コネクションおよびデータベースサーバのテーブル/レコードを解放できます。

Java EE Server リクエストタイムアウト＝業務内で異常な遅延が発生していると判断する時間

タイムアウトパラメーター間の関係

遅延が発生した部位の絞り込みを容易にするため、またタイムアウトパラメーター間の矛盾がないように、バックエンドに近づくほどタイマーを小さく設定します。

Web Server プロキシータイムアウト＞Java EE Server リクエストタイムアウト＞トランザクションタイムアウト

2.4.4 パラメーター一覧

タイムアウトのパラメーターは、asadmin の get サブコマンドで設定値を確認し、asadmin の set サブコマンドで設定値を変更できます。
また、一部のパラメーターは、サーバテンプレートを書き換えることで設定値を変更します。
パラメーター一覧を次に示します（Java EE サーバの名称が JavaEE1 の場合）。

表 2.4-2 タイムアウト パラメーター一覧

#	定義項目（初期値：″=xx″）	説明
1	サーバテンプレート reverse_proxy.conf@.vtl の ProxyTimeout ディレクティブの値	Web Server プロキシータイムアウト（単位：秒）。1～65535 の間で指定します。
2	configs.config.*JavaEE1*-config.transaction-service.timeout-in-seconds=180	トランザクションタイムアウト（単位：秒）。0～2147483647 の間で指定します。0 を指定した場合は、タイムアウトしません。
3	configs.config.*JavaEE1*.network-config.protocols.protocol.http-listener-1.http.request-timeout-seconds=190	Java EE Server リクエストタイムアウト（単位 : 秒）。0～2147483647 の間で指定します。0 を指定した場合は、タイムアウトしません。

2.5 タイムアウト（HiRDB）

2.5.1 設計目的

Java EE サーバおよびデータベースサーバで、処理遅延、通信遅延の障害が発生した場合、次のような問題が発生します。

- 業務処理が滞留し、その業務処理がスレッドやテーブル/レコードなどのリソースを占有し、ほかの業務処理に影響を与える
- 利用者を不要に待たせる

このような問題が発生する場合、システムでは次の対処を実施する必要があります。

- 遅延を検知
- 業務処理が占有しているリソースを解放
- 利用者へのエラー応答

タイムアウトパラメーターは、遅延を検知し、リソースの解放、利用者へのエラー応答をするためのパラメーターです。このパラメーターを設定して、処理遅延や通信遅延の障害が発生した場合でも、システムや利用者への影響を軽減するようにします。

2.5.2 設計パラメーター

タイムアウトパラメーターは 5 つあります。遅延の検知を複数の個所で実施することで、遅延が発生した個所の絞り込みが容易になります。

各タイムアウトパラメーターの説明と、それぞれのタイムアウト時の動作を次に示します。

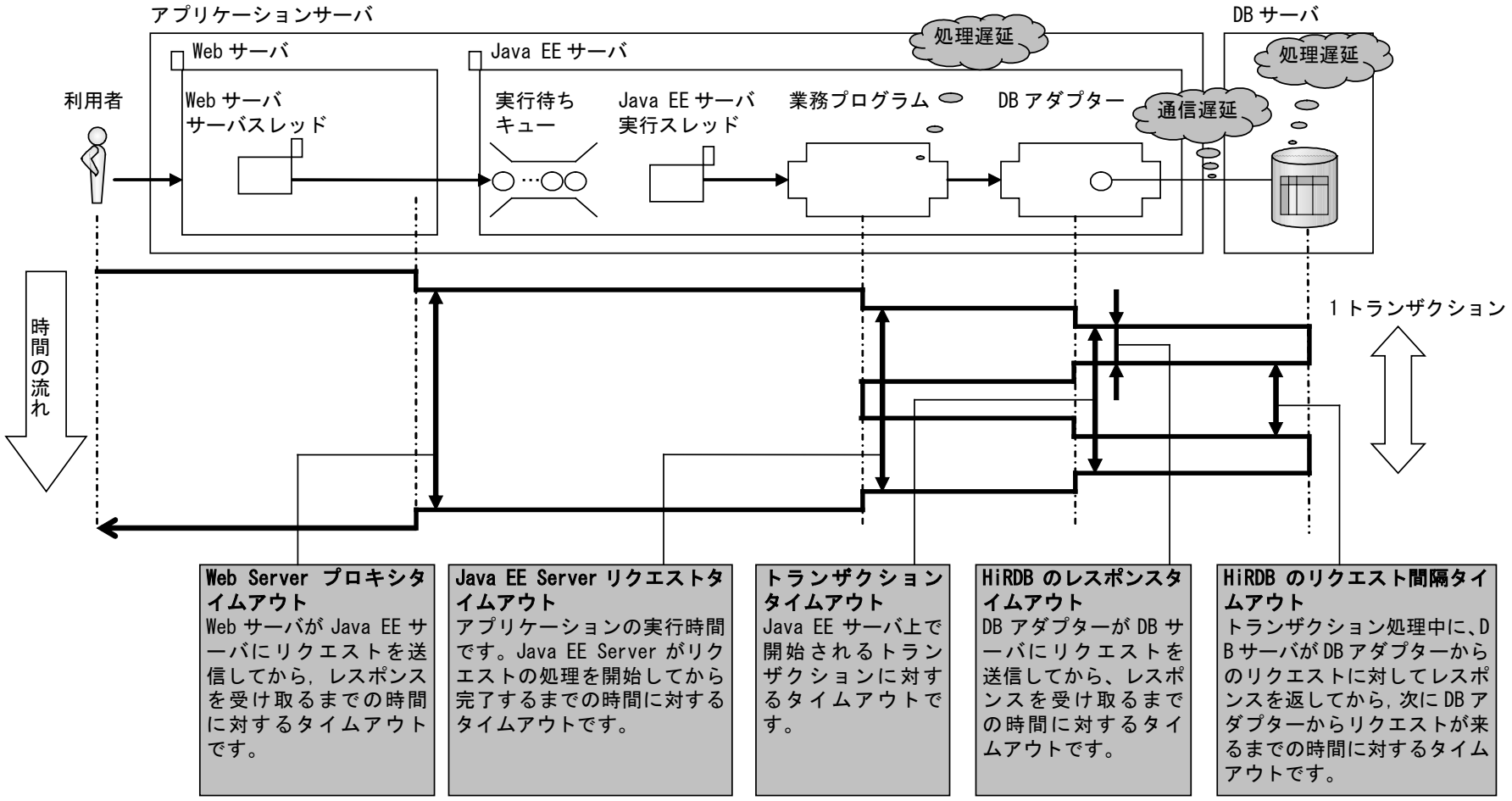

図 2.5-1 タイムアウトパラメーター

表 2.5-1 タイムアウト時の動作

#	パラメーター	利用者へのエラー応答	リソース解放	ログ
1	Web Server プロキシータイムアウト	HTTP ステータスコード 502（Proxy Error）を返します。	Web サーバのサーバスレッド、コネクションを解放します。(ただし､Java EE サーバ内の処理は継続します。)	Web サーバのログにメッセージ(KH00363 または KH00364)を出力。ただし､メッセージに含まれる接続先が Java EE サーバである場合。
2	Java EE Server リクエストタイムアウト	なし(アプリケーションの実行スレッドに対して割り込みが発生します。具体的には、Java EE Server が実行スレッドに対して、Thread#interrupt()メソッドを呼び出します。そのため，Java EE Server リクエストタイムアウトが発生するとアプリケーションで例外が発生することがあります。interrupt()メソッドについては、javadoc を参照してください。)。	Java EE サーバのスレッド、JDBC コネクションおよびデータベースサーバのテーブル/レコードを解放します。	－
3	トランザクションタイムアウト	なし（JTA インタフェースを利用してトランザクション制御している場合はアプリケーションに例外が返ります)。	－	－
4	HiRDB のレスポンスタイムアウト	なし（アプリケーションに例外が返ります)。	アプリケーションに例外を返し、テーブル/レコード、コネクションを解放します。	DB アダプターのメッセージログにメッセージ(KDJE50036-E)を出力。
5	HiRDB のリクエスト間隔タイムアウト	なし（アプリケーションに例外が返ります)。	Java EE サーバに異常が発生したと見なし、HiRDB がコネクションを切断します。また、トランザクションをロールバックし、テーブル/レコードを解放します。	－

（凡例） －：該当なし

2.5.3 設計指針

<u>利用者へのエラー応答を目的としたパラメーター</u>

●Web Server プロキシータイムアウト

Web サーバ以降（Java EE サーバ、およびデータベースサーバ）の処理で、遅延障害が発生していると判断する時間を設定します。

Web Server プロキシータイムアウト＝Web サーバ以降の処理で遅延障害が発生していると判断する時間

また、実行待ちキューサイズの設計時に考慮した、「許容できる内部保留時間」よりも大きい値を設定してください。小さい値を設定すると、キューに格納したリクエストの処理終了を待たないで、Web サーバでタイムアウトする場合があります。

Web Server プロキシータイムアウト＞許容できる内部保留時間＝内部保留時間＋実行待ちキュー内の保留時間

●トランザクションタイムアウト

全トランザクションの処理時間を踏まえ、明らかにトランザクション処理中に異常な遅延が発生していると判断する時間を設定します。

トランザクションタイムアウト＝トランザクション処理中に異常な遅延が発生していると判断する時間

リソースの解放を目的としたパラメーター

●Java EE Server リクエストタイムアウト

全業務の処理時間を踏まえ、明らかに業務内で異常な遅延が発生していると判断する時間を設定します。これによって、Java EE サーバのスレッド、JDBC コネクションおよびデータベースサーバのテーブル/レコードを解放できます。

Java EE Server リクエストタイムアウト＝業務内で異常な遅延が発生していると判断する時間

●HiRDB のレスポンスタイムアウト

全 SQL 処理の処理時間を踏まえ、明らかに SQL 処理中に異常な遅延が発生していると判断する時間を設定します。これによって、Java EE サーバの JDBC コネクションおよびデータベースサーバのテーブル/レコードを解放できます。

HiRDB のレスポンスタイムアウト＝SQL 処理中に異常な遅延が発生していると判断する時間

●HiRDB のリクエスト間隔タイムアウト

Java EE サーバがトランザクションタイムアウトで回避できない障害に陥っていると判断する時間として、トランザクションタイムアウトと同じ値を設定します。これによって、データベースサーバのコネクション/テーブル/レコードを解放できます。

HiRDB のリクエスト間隔タイムアウト＝トランザクションタイムアウト

タイムアウトパラメーター間の関係

遅延が発生した部位の絞り込みを容易にするため、またタイムアウトパラメーター間の矛盾がないように、バックエンドに近づくほどタイマーを小さく設定します。

Web Server プロキシータイムアウト＞Java EE Server リクエストタイムアウト＞トランザクションタイムアウト＞HiRDB のレスポンスタイムアウト

2.5.4 パラメーター一覧

タイムアウトのパラメーターは、asadmin の get サブコマンドで設定値を確認し、asadmin の set サブコマンドで設定値を変更できます。
また、一部のパラメーターは、サーバテンプレートを書き換えることで設定値を変更します。
パラメーター一覧を次に示します（Java EE サーバの名称が JavaEE1 の場合）。

表 2.5-2 タイムアウト パラメーター一覧

#	定義項目（初期値：″=xx″）	説明
1	サーバテンプレート reverse_proxy.conf@.vtl の ProxyTimeout ディレクティブの値	Web Server プロキシータイムアウト（単位：秒）。1～65535 の間で指定します。
2	configs.config.*JavaEE1*-config.transaction-service.timeout-in-seconds=180	トランザクションタイムアウト（単位：秒）。0～2147483647 の間で指定します。0 を指定した場合は、タイムアウトしません。
3	configs.config.*JavaEE1*.network-config.protocols.protocol.http-listener-1.http.request-timeout-seconds=190	Java EE Server リクエストタイムアウト（単位：秒）。0～2147483647 の間で指定します。0 を指定した場合は、タイムアウトしません。
4	HiRDB.ini ファイル： [HIRDB] PDCWAITTIME=0※	HiRDB のレスポンスタイムアウト（単位：秒）。0～65535 の間で指定します。0 を指定した場合は、タイムアウトしません。
5	HiRDB.ini ファイル： [HIRDB] PDSWAITTIME=600※	HiRDB のリクエスト間隔タイムアウト（単位：秒）。0～65535 の間で指定します。0 を指定した場合は、タイムアウトしません。

注※　asadmin の create-jdbc-connection-pool サブコマンドの--property で、コネクションプール作成時に指定してください。

2.6 利用者・サーバ間の通信障害に関するタイムアウト

2.6.1 設計目的

利用者・サーバ間で通信障害が発生した場合、次のような問題が発生します。

- Web サーバのサーバスレッドが占有され、リクエスト処理ができなくなる

このような問題が発生する場合、システムでは次の対処を実施する必要があります。

- 通信障害を検知
- 占有しているリソースを解放
- 利用者へのエラー応答

利用者・サーバ間の通信障害に関するタイムアウトパラメーターは、通信障害を検知し、リソースの解放、利用者へのエラー応答をするためのパラメーターです。具体的には次の場合にタイムアウトします。

- 利用者からのリクエスト受信（コネクション確立後、HTTP プロトコルの受信）中にデータを受信しなくなった場合
- 利用者へのレスポンス送信中にデータを送信できなくなった場合

このパラメーターを設定して、障害が発生した場合でも、システムや利用者への影響を軽減するようにします。

2.6.2 設計指針

明らかに利用者・サーバ間に通信障害が発生していると判断する時間を設定します。これによって、Web サーバのスレッドを解放できます。

利用者・サーバ間の通信障害に関するタイムアウト＝利用者・サーバ間に通信障害が発生していると判断する時間

2.6.3 パラメーター一覧

パラメーター一覧を次に示します。

表 2.6-1 利用者･サーバ間の通信障害に関するタイムアウト パラメーター一覧

#	定義項目（初期値）	説明
1	サーバテンプレート httpsd.conf@windows.vtl の Timeout ディレクティブの値を直接書き換えます。	Web Server タイムアウト（単位：秒）。0～65535 の間で指定します。0 を指定した場合は、待ち時間が 0 秒になります。。

3 システム要件の検証

サイジングでは、システム要件と処理性能から必要なマシン構成（マシンスペックとマシン台数）を見積もりました。また、2 章では性能・信頼性確保のためのチューニングパラメーターを設計しました。この章では、これらの設計の結果、システムがシステム要件を満たせているかどうかを検証します。

本章の構成

3.1 検証手順

システム要件の検証の流れを次に示します。

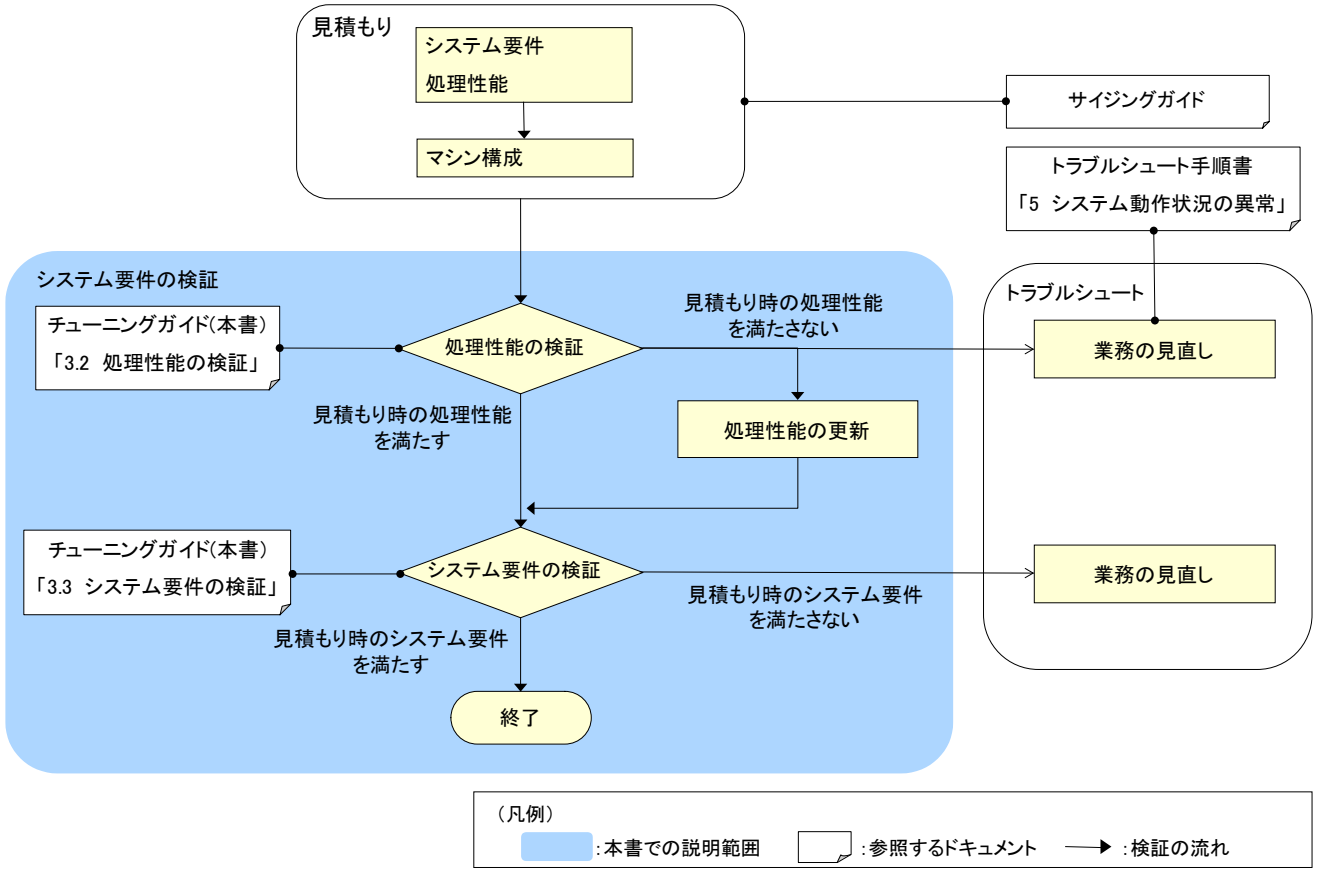

図 3.1-1 システム要件の検証の流れ

3.2 処理性能の検証

見積もり時の処理性能を満たしているかどうかを検証します。

3.2.1 検証対象

次の処理性能を確認します。

表 3.2-1 検証対象処理性能

#	検証対象	確認方法※
1	CPU 使用率	稼働情報ファイル （詳細については、「3.2.2 稼働分析方法」を参照）
2	内部保留時間	同上
3	使用 Survivor 領域サイズ （＝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）	同上
4	使用 Tenured 領域サイズ （＝アプリケーション使用 Tenured 領域サイズ＋300MB）	同上
5	1 セッション当たりの使用メモリーサイズ	同上

注※　サーバ起動直後は、コネクション生成などのコストなどによってレスポンス時間が遅くなる可能性があるため、ある程度時間が経過し、測定値が一定になっているところを採用します。

3.2.2 稼働分析方法

次のファイルを、アプリケーションサーバとは別の安全なマシンに格納し管理します。すべてのアプリケーションサーバに対して実施します。

表 3.2-2 バックアップ対象の稼働統計ファイル

#	分類	ファイル
1	Web サーバ履歴	C:¥Hitachi¥APServer¥javaee¥logs¥nodes¥ localhost-domain1¥Web1¥access.* （access で始まるすべてのファイル）
2	J2EE サーバ履歴	C:¥Hitachi¥APServer¥javaee¥logs¥nodes¥localhost-domain1¥JavaEE1¥statistics¥* （statistics ディレクトリー以下のすべてのファイル）
3		C:¥Hitachi¥APServer¥javaee¥glassfish¥nodes¥localhost-domain1¥JavaEE1¥config¥javalog<nn>.log （javalog で始まるすべてのファイル）
4	OS 履歴	OS 稼働情報ファイル

表 3.2-2 でバックアップした稼働情報を使用して、システムの利用状況の変化を確認し、サイジングを見直します。稼働情報の詳細は、表 3.2-4 から表 3.2-9 を参照してください。使用する稼働情報を次に示します。

表 3.2-3 システム利用状況の確認

#	分析対象	稼働情報	
		ファイル	項目
1	CPU 使用率	OS 稼働統計ファイル	－
2	内部保留時間	access.*	%r %T
3	使用 Survivor 領域サイズ	javalog<nn>.log	survivor_info
4	使用 Tenured 領域サイズ	javalog<nn>.log	tenured_info
5	1 セッション当たりの使用メモリーサイズ	JVMMemoryExtensionsStatisticsJavaEE1_ <YYYYMMDDhhmm> <TZ>.csv	httpsessionexplicitmemoryblockmaxsize

（凡例） －：該当なし

表 3.2-4 Web サーバ履歴の確認項目

ファイル名	access.*
フォーマット	%h %l %u %t ¥"%r¥" %>s %b %T %{hws_thread_id}P %{hws_ap_root}n
例 [カラム]	10.100.10.90 - - [29/Aug/2014:17:10:06.718 +0900] "GET /test/Welcome.do HTTP/1.1" 200 680 [1] [4] [5] [6] [7] 0.015 2136 10.100.10.80/2932/0x0000000000000001 [8] [9] [10]
%h	[1]クライアントのホスト名
%l	－
%u	－
%t	**[4]リクエスト処理を開始した時刻**
%r	**[5]HTTP 通信のリクエストの先頭行**
%>s	[6]最終ステータス
%b	[7]送信バイト数（HTTP ヘッダー、および chunked エンコーディングによって追加されたデータを除く）
%T	**[8]リクエスト処理にかかった時間（秒）**
%{hws_ thread _id}P	[9]HTTP 通信のリクエストを処理するスレッド ID
%{hws_ap_root}n	[10]ルートアプリケーション情報

（凡例） **太字**：表 3.2-3 および表 3.3-2 で確認する項目です。

表 3.2-5 JavaVM ログの確認項目

ファイル名	javalog<nn>.log
フォーマット	id,date,gc_kind,gc_info,gc_time,eden_info,survivor_info,tenured_info,perm_info,cause_info,user_cpu,system_cpu
例 [カラム]	VGC,Tue Feb 16 16:18:40.734 2010,GC,,,,0,383,17472,0,0,349568,,,, [1] [2] [3] [12] [15]
id	[1]JavaVM ログファイル識別子（VGC: 拡張 verbosegc 情報）
date	[2]稼働情報を出力した時刻
gc_kind	[3]GC 種別（Full GC または GC）
survivor_info <GC 後の領域長>	**[12]使用 Survivor 領域サイズ**
tenured_info <GC 後の領域長>	**[15]使用 Tenured 領域サイズ**

（凡例） **太字**：表 3.2-3 で確認する項目です。

表 3.2-6 JavaVM 履歴の確認項目

ファイル名	JVMMemoryExtensionsStatisticsJavaEE1 _<YYYYMMDDhhmm><TZ>.csv
フォーマット	Date(+0900),,,,,,,ExplicitHeapSize.Count,,ExplicitMemoryBlockMaxSize.Count,,HttpSessionExplicitMemoryBlockMaxSize.Count,,,,,,,,,,
例 [カラム]	2014/08/22 16:05:38.081,,,,,,,65536,,0,,0,,,,,,,,,, [1] [8] [10] [12]
Date(+0900)	[1]稼働情報を出力した時刻
ExplicitHeapSize.Count	[8] Explicit ヒープサイズ[バイト]
ExplicitMemoryBlockMaxSize.Count	[10]過去 60 秒間の Explicit メモリーブロックの最大サイズ[バイト]
HttpSessionExplicitMemoryBlockMaxSize.Count	**[12]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[バイト]**

（凡例） **太字**：表 3.2-3 で確認する項目です。

表 3.2-7 スレッドプール履歴の確認項目

ファイル名	ThreadPoolStatistics_JavaEE1_<YYYYMMDDhhmm><TZ>.csv
フォーマット	Date,ObjectName,,,ActiveThreadCount.UpperBound,,ActiveThreadCount.HighWaterMark,,,, WaitingRequestCount.UpperBound,,WaitingRequestCount.HighWaterMark,,,, OverflowRequestCount.Count,,,,,,SessionCount.UpperBound,,SessionCount.HighWaterMark,,
例 [カラム]	2007/12/28 10:39:54.646,"XXX,name=Struts Test Application",,,5,,5,,,,10,,1,,,,0,,,,,,300,,220,, [1] [2] [5] [7] [11][13][17] [23] [25]
Date(+0900)	[1]稼働情報を出力した時刻
ActiveThreadCount. UpperBound	[5]同時実行スレッド数の上限値
ActiveThreadCount. HighWaterMark	[7]過去 60 秒間の最大同時実行スレッド数

表 3.2-8　Web リクエスト稼働履歴の確認項目

ファイル名	WebRequestStatistics_JavaEE1_<YYYYMMDDhhmm><TZ>.csv
フォーマット	Date,ObjectName,,,ActiveThreadCount.UpperBound,,ActiveThreadCount.HighWaterMark,,,, WaitingRequestCount.UpperBound,,WaitingRequestCount.HighWaterMark,,,, OverflowRequestCount.Count,,,,,,SessionCount.UpperBound,,SessionCount.HighWaterMark,,
例 [カラム]	2007/12/28 10:39:54.646,"XXX,name=Struts Test Application",,,5,,5,,,,10,,1,,,,0,,,,,,300,,220,, [1] [2] [5] [7] [11][13][17] [23] [25]
Date(+0900)	[1]稼働情報を出力した時刻
WaitingRequestCount. UpperBound	[11]実行待ちリクエスト数の上限値
WaitingRequestCount. HighWaterMark	[13]過去 60 秒間の最大実行待ちリクエスト数
OverflowRequestCount. Count	[17]実行待ちリクエスト数の上限からあふれた累積リクエスト数

表 3.2-9 セッション稼働履歴の確認項目

ファイル名	WebSessionStatistics_JavaEE1_<YYYYMMDDhhmm><TZ>.csv
フォーマット	Date,ObjectName,,,ActiveThreadCount.UpperBound,,ActiveThreadCount.HighWaterMark,,,, WaitingRequestCount.UpperBound,,WaitingRequestCount.HighWaterMark,,,, OverflowRequestCount.Count,,,,,,SessionCount.UpperBound,,SessionCount.HighWaterMark,,
例 [カラム]	2007/12/28 10:39:54.646,"XXX,name=Struts Test Application",,,5,,5,,,,10,,1,,,,0,,,,,,300,,220,, [1] [2] [5] [7] [11][13][17] [23] [25]
Date(+0900)	[1]稼働情報を出力した時刻
SessionCount.UpperBound	[23]セッション数の上限値
SessionCount.HighWaterMark	**[25]過去 60 秒間のセッション数の最大値**

（凡例）　**太字**：表 3.3-2 で確認する項目です。

3.2.3 検証方法

＜手順＞

① **負荷ツールなどを使用し、システム要件の 1 秒当たりのリクエスト数・同時ログイン数を再現する**

ただし、実運用時よりも少ないマシン台数で実行する場合は、マシン台数に応じたリクエスト数・同時ログイン数にしてください。

② **①をメンテナンス間隔中に想定される分量だけ実施する**

例えば、1 か月おきにアプリケーションサーバを再起動する場合、1 か月分のリクエストを実施します。

3.2.4 検証結果

検証の結果、処理性能を満たしているかどうかを確認します。満たせていない場合、次のどちらかの対策を実施します。

- 業務の見直し

  業務の冗長な処理（I/O 回数、SQL 発行回数、データ転送量、メモリー使用量が多いなど）を見直すことで処理性能を改善します。

  トラブルシュート手順書を参考に、業務を見直します。
- 処理性能の更新

  業務の見直しが困難な場合、測定した処理性能を新たに採用し、サイジング・チューニングを再度実施します。

3.3 システム要件の検証

見積もり時のシステム要件を満たしているかどうかを検証します。

3.3.1 検証対象

次のシステム要件を確認します。

表 3.3-1 検証対象システム要件

#	検証対象	確認方法※
1	1秒当たりのリクエスト数	稼働情報ファイル （詳細については、表 3.3-2 を参照）
2	同時ログイン数	同上
3	目標レスポンス時間	負荷ツール

注※　サーバ起動直後は、コネクション生成などのコストなどによってレスポンス時間が遅くなる可能性があるため、ある程度時間が経過し、測定値が一定になっているところを採用します。

表 3.3-2 システム利用状況の確認

#	分析対象	稼働情報	
		ファイル※	項目
1	1 秒当たりのリクエスト数	access.*	%t %r
2	同時ログイン数	WebSessionStatistics_JavaEE1_<YYYYMMDDhhmm><TZ>.csv	SessionCount.HighWaterMark

注※　ファイルの詳細については、「3.2.2　稼働分析方法」を参照してください。

3.3.2 検証方法

＜手順＞

① **負荷ツールなどを使用し、システム要件の１秒当たりのリクエスト数・同時ログイン数を再現する**

ただし、実運用時よりも少ないマシン台数で実行する場合は、マシン台数に応じたリクエスト数・同時ログイン数にしてください。

② **①をメンテナンス間隔中に想定される分量だけ実施する**

例えば、1 か月おきにアプリケーションサーバを再起動する場合、1 か月分のリクエストを実施します。

3.3.3 検証結果

検証の結果、システム要件を満たしているどうかか確認します。満たせていない場合、次の対策を実施します。

- 業務の見直し

  業務の排他処理などを見直すことでシステム要件を満たすようにします。