WebOTX
Standard/Enterprise Edition V5
のチューニング方法

2006 年 11 月 8 日
第 1 版

NEC
第二システムソフトウェア事業部
AP サーバ開発グループ

目次

１．　初めに

本資料(チューニング～WebOTX Standard/Enterprise EditionV5 のチューニング方法～)では、WebOTX　Standard Edition V5 または Enterprise Edition V5 を使用している環境において、チューニングの考え方やその設定方法について説明しています。

２．　機能(チューニング関連)

この章では WebOTX のチューニングポイント及びその方法について記載しています。

2.1 アプリケーショングルーピング

WebOTX Standard/Enterprise Edition ではサーバアプリケーションは「アプリケーショングループ」および「プロセスグループ」という単位でグルーピングを行ないます。それぞれどのような観点でグルーピングを行なうべきか説明します。

2.1.1 アプリケーショングループ

アプリケーショングループとは、開始・終了などの運用を共にするアプリケーション群をグルーピングしたものです。例えば、「受発注業務」「在庫数管理業務」といったような業務毎にツリーを分けて管理するといった方法を用います。こうすることで、「受発注業務」は 10 時から 17 時まで動作させ、その後は停止させる。「在庫数管理業務」は 24 時間動作させるなどのアプリケーショングループ単位での独立した運用が行えるようになります。なおアプリケーションに関する設定(例えば、プロセスグループのプロセス数、スレッド数、Java VM オプション等)を変更する場合はアプリケーショングループの再起動が必要となります。

すなわち、業務運用の単位でアプリケーショングループをグルーピングします。

2.1.2 プロセスグループ

プロセスグループとは、ある特定のサービスを提供するプロセス群です。同一のプロセスグループに登録されているコンポーネントは同一のプロセス上でロードされます。WebOTX はビジネスロジックを記述した 1 つまたは複数のコンポーネントをプロセスグループに登録し、プロセスとして実行します。

プロセスグループの単位で次の項目が共有されます。

・キュー(受信用)

WebOTX ではプロセスグループ単位にキューを生成します。同一プロセスグループ上のリクエストはすべてこのキューにキューイングされます。

・Java VM(Java 関連のプロセスグループのみ)

WebOTX ではプロセスグループ単位に Java VM を生成し、配下のコンポーネントをロードします。マルチプロセス構成の場合は同一構成の Java VM が複数生成されます。

Java VM に関する設定(ヒープサイズのような VM オプション等)もプロセスグループ単位での設定となります。

・実行多重度

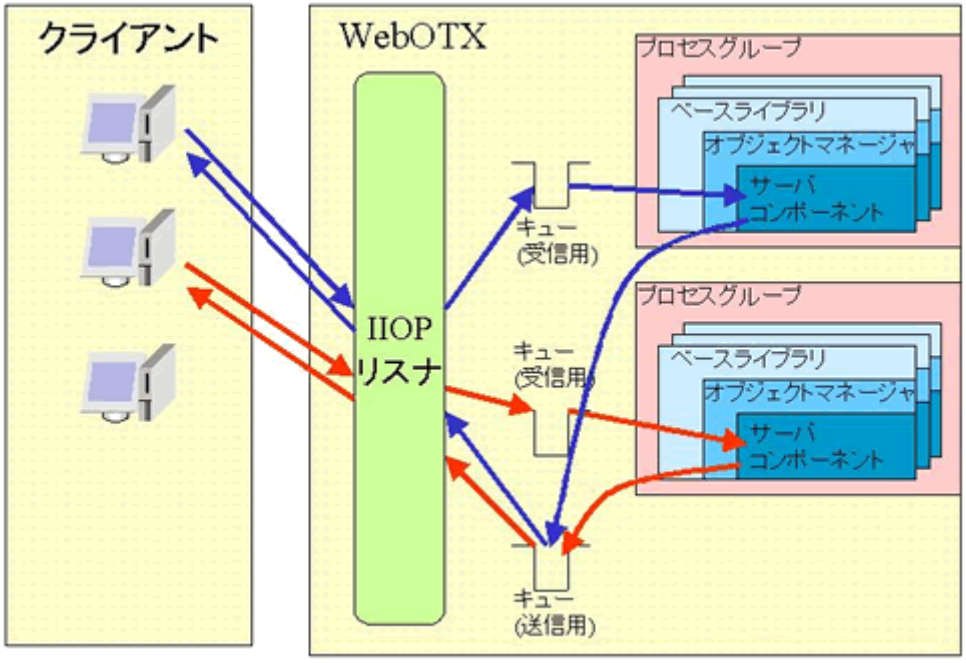
WebOTX ではプロセスグループ単位に実行多重度(プロセス、スレッド)を設定します。そのプロセスグループの特性に応じて実行多重度を設定できます。

・バックエンドサーバへのコネクション

AP サーバが利用するバックエンドサーバ(DB や ACOS や TPBASE)へのコネクションはプロセスで共有されます。よってプロセスグループを分けたり、マルチプロセスにしたりする場合はその分コネクションが生成されることになります。また VIS コネクタの場合、全てのプロセスグループで定義している総スレッド数分のコネクションが生成されます。

2.1.3 処理の流れ

Standard/Enterprise Edition のサーバ AP での処理の流れについて説明します。



上記で説明したように、プロセスグループ単位でキュー(受信用)が生成されます。クライアント(thin クライアントの構成では Web サーバ)からの要求は IIOP リスナを経由して、該当プロセスグループのキューにキューイングされます。そのときにプロセスグループ内でアイドル中(フリーな)のスレッドが存在していた場合、即時にそのスレッドでリクエストは実行されます。アイドル中のスレッドが無い場合は、キューで実行待ちとなります。

プロセスグループでキューは共有しますので、例えば実行時間が非常に長い呼び出しが

実行された場合、そのリクエストを処理するために 1 つのスレッドは占有されます。これにより同じプロセスグループ内にある他の呼び出しを実行するスレッドが減少してしまいます。全てのスレッドが長い呼び出しに占有されてしまった場合、そのプロセスグループに対する要求は全てキューイングされることになります。

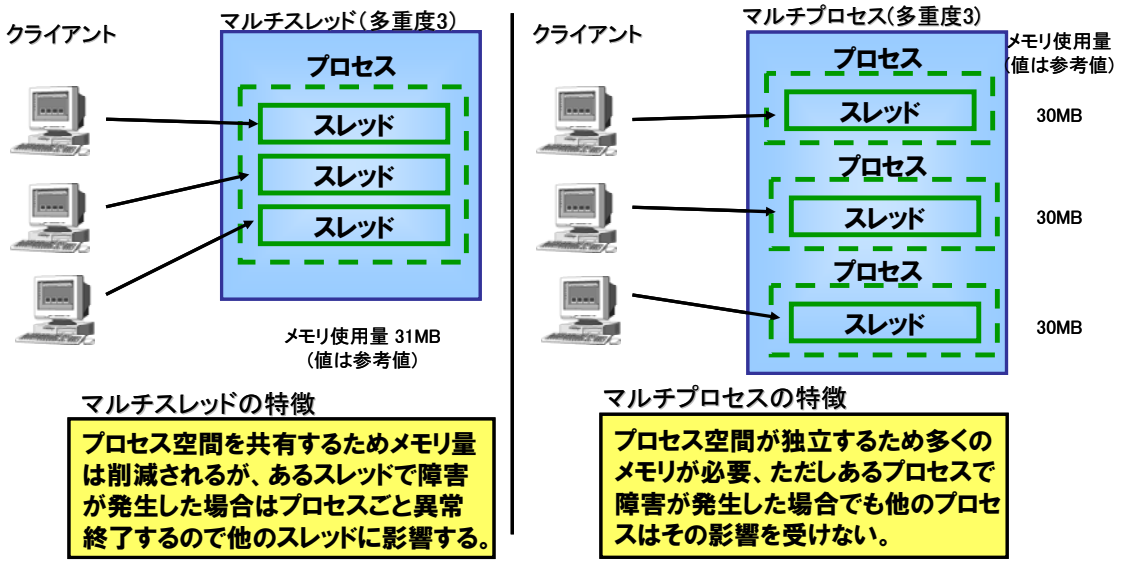
2.1.4 マルチプロセス・マルチスレッド

プロセスグループを多重実行させる場合、プロセスを多重化させる方法とスレッドを多重化させる方法があります。

各プロセスはシングルスレッドでもマルチスレッドでも動作させることができます。すべてのプロセス上のスレッド群は、単一のキューに対してデータの到着を待ち合わせるので、空きスレッドに効率良くトランザクション要求をディスパッチすることができます。マルチプロセス実行させることにより、アプリケーションの一時的な障害に対してサービスの継続が可能になります。データや環境、タイミングなどの要因でアプリケーション障害が発生しても、そのプロセスだけが異常終了し、他のプロセスは影響を受けません。したがって、システム全体ではサービスを中断させることがありません。

しかしマルチプロセス構成はマルチスレッド構成に比べてより多くのリソースを必要とします。システムで多数のプロセスが起動することによりマシンのリソース不足を招き、サーバ自体が不安定になる恐れがあります。マルチプロセスは予期せぬアプリケーション障害に備えての多重化とし、クライアントからの同時実行に備えての多重化はできるだけマルチスレッドで行う構成が望ましいといえます。ただし、アプリケーションの構成上(スレッド間同期、排他制御)の問題でマルチスレッド動作できないなどの制限がある場合は、マルチプロセス構成で多重度を確保します。

また、システム運用中に発生した想定外の大量のトランザクション要求に対して、動的にアプリケーション多重度(スレッド、プロセス)を増加させることが可能です（注：スレッド数は最初に確保した数以上は増やせません)。これにより、大量のトランザクション要求に対してサーバ側が処理しきれずに要求がキューに滞留しレスポンス悪化することが避けられます。また、負荷が下がったときに安全に実行多重度を元に戻すこともできます。



2.1.5 グルーピングのポイント

上記の WebOTX のプロセスグループの特性を考慮してどのようにグルーピングするのが望ましいか説明します。

消費メモリについて

多重度を確保するにはマルチプロセスとマルチスレッドがありますが、特にマルチプロセス構成にする場合メモリ消費量を考慮する必要があります。以下に WebOTX Java AP のメモリ消費量の目安の計算式を示します。

(ヒープサイズ)+(スレッドスタックサイズ)×スレッド数＋１５(MB)

※1 プロセスあたりの計算式です。実際は全てのプロセスの合計値となります。
スレッド数を上げても(スレッドスタックサイズ:既定値 1M)分しか増加しないのに比べ、プロセス数を上げると上記の計算式の値全てのサイズ分増加します。メモリ量を考慮すると、多重度はマルチプロセスよりマルチスレッドで行うべきです。

ライセンスについて

VIS コネクタを使用した場合、WebOTX システムで定義している総スレッド数分コネクションを作成します。VIS コネクタのライセンスはコネクション単位ですので、総スレッド数がライセンスを越えることができません。

例えば

WebOTX VIS コネクタ実行環境（4）

WebOTX VIS コネクタ実行環境 (+8)

を購入している場合 4+8(=12)ライセンスですので、1 プロセスグループ 1 プロセス 1 スレッド構成にするにしても 12 個しかプロセスグループを作成することができません。1 プロセスグループ 2 プロセス 2 スレッド構成の場合は 3 個(12÷4)のプロセスグループしか作成することができないこととなります。

コネクション数について

バックエンドサーバへのコネクション数についても注意が必要となります。マルチプロセス構成とした場合、プロセスの数分、コネクションを生成することとなります。多くのプロセスを生成すると、バックエンドサーバとのコネクション数の制限を越えてしまうことがあります。

キューについて

上記にも書きましたように、WebOTX はプロセスグループ単位でキューを生成します。ここで作成したコンポーネントを複数のプロセスグループに登録するのと、1 プロセスグループに登録するのではどちらがいいのかの考え方について説明します。

・複数のプロセスグループに登録する場合

複数のプロセスグループに登録する場合は、キューやプロセス空間が他のコンポーネントと完全に独立します。あるコンポーネントに問題があり、ストールやレスポンス悪化、アボートが発生した場合でもその影響をほとんど受けません。あるコンポーネントがストールもしくはレスポンス悪化(もともと正常でも時間がかかる呼び出しでも同様)があった場合、プロセスグループ中のスレッドが、全てストールしてしまう可能性があります。アボートの場合は、Java VM 全体でアボートとなり、全てのコンポーネントへの影響が出てきます。このような状態になった場合、そのプロセスグループへの呼び出しはその呼び出し自体に問題が無くてもそれ以上処理されなくなりそのプロセスグループ全体がストールしたり、実行プロセスがアボートしてなくなってしまいます。プロセスグループを分けることは、このような現象を未然に予防することができます。

・1 プロセスグループに登録する場合

上記で説明したように 1 プロセスグループに登録する場合ストール、レスポンス悪化、アボートに対して注意が必要になります。対策として、マルチスレッド構成にして多重度を十分に確保すること、あるいはマルチプロセス構成にしてアボートしてもプロセス数が 0 とならないようにしておくことがあげられます。複数のプロセスグループの構成より万全ではありませんがある程度問題は回避できます。

2.2 タイマ設定

高負荷になりレスポンスが悪化した場合などに考慮が必要なタイムアウト値や、プロセス起動・停止に必要なタイムアウト値などに関する設定、その他上限設定について説明します。

2.2.1 起動・停止タイムアウト(メッセージ表示)

プロセスの起動操作、停止操作の完了を待ち合わせる時間を設定します。

タイムアウトした場合はエラーメッセージが表示されます。但し、プロセスの起動処理自体は実行され続けます。

設定項目	説明	既定値	設定値範囲
1. システム起動タイムアウト	システムの起動処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位
2. システム停止タイムアウト	システムの停止処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位
3. アプリケーショングループ起動タイムアウト	アプリケーショングループの起動処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位
4. アプリケーショングループ停止タイムアウト	アプリケーショングループの停止処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位
5. プロセスグループ起動タイムアウト	プロセスグループの起動処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位
6. プロセスグループ停止タイムアウト	プロセスグループの停止処理に関するタイムアウトの設定をします。	60 秒	1 以上の整数で秒単位

2.2.2 起動・停止タイムアウト(プロセス終了)

アプリケーションプロセスの起動または停止に長い時間がかかる場合はタイムアウトを検出して異常終了することがあります。この場合、処理時間が妥当か検証し、妥当でないなら処理を見直し、妥当ならタイマ値を見直す必要があります。

設定項目	説明	既定値	設定値範囲
1. スレッド初期化時間	スレッド初期化にかかる時間のタイムアウト値を設定します。	600 秒	1 以上の整数で秒単位

スレッド初期化時間のタイムアウト値は以下の場所で時間のかかる処理を行っている場合に考慮が必要です。

- コンストラクタ・デストラクタ(アパートメントの場合)
- 常駐オブジェクトのコンストラクタ・デストラクタ

設定を反映させるためにはアプリケーショングループを再起動してください。

設定方法

運用管理ツールのプロセスグループのプロパティ
[スレッド制御]-[スレッド初期化時間]

2.2.3 呼び出しタイムアウト

- クライアント側で設定する場合

クライアント側でオペレーション実行要求を出してからその応答をもらうまでのタイムアウト値を設定します。このタイムアウト値を設定することで、クライアントが無応答となることを抑止します。このタイムアウト値はサーバでの実行時間のほかに通信に要する時間およびキュー待ち時間を考慮して設定します。

設定項目	説明	既定値	設定値範囲
1. Java クライアントからのオペレーション呼び出しタイムアウト時間	EJB、JNDI など RMI-IIOP 通信を使用する呼び出しのタイムアウト時間です。	無制限	0 以上 ※0 を指定した場合は無制限
2. Java クライアントでの無通信監視タイムアウト時間	クライアント側で一定時間送受信要求のないコネクションをクローズするまでのタイムアウト時間です。	無制限	0 以上 ※0 を指定した場合は無制限
3. オペレーション呼び出しのタイムアウト時間	CORBA 通信を使用する呼び出しのタイムアウト時間です。	30 秒	0 以上 ※0 を指定した場合は無制限
4. コネクションラウンドロビンでのオペレーション呼び出しのタイムアウト時間	多重化オブジェクトによるコネクションラウンドロビン機能を利用する場合に有効になります。 個々のサーバへのオペレーション呼び出しのタイムアウト時間です。	オペレーション呼び出しのタイムアウト時間の値	0 以上 ※0 を指定した場合は無制限

設定方法

1. Java クライアントからのオペレーション呼び出しタイムアウト時間

・ Java コマンドラインで指定する場合
以下の形式でシステムプロパティを指定
-Djp.co.nec.orb.ClientRequestTimeout=<タイムアウト時間(秒)>
・ J2EE サーバ(Web コンテナ)に指定する場合
Web コンテナ運用管理コンソールの「Java 設定」-「Java VM の引数」に以下の形式でシステムプロパティを追加
-Djp.co.nec.orb.ClientRequestTimeout=<タイムアウト時間(秒)>
・ J2EE サーバ(JNDI サーバ)に指定する場合
JNDI サーバの構成情報ファイルの javaargs(UNIX の場合は USER_JAVAARGS)のキーに以下の形式でシステムプロパティを追加
-Djp.co.nec.orb.ClientRequestTimeout=<タイムアウト時間(秒)>
注）JNDI サーバの構成情報ファイル
Windows　%JNDI_HOME%¥config¥jndiserver.conf
UNIX　/etc/WebOTX/ee/jndi.conf
・ プロセスグループ(EJB コンテナ)に指定する場合：
運用管理ツールのプロセスグループを選択し、「設定」-「プロパティ」を選択する。「Java システムプロパティ」タブで、プロパティに「jp.co.nec.orb.ClientRequestTimeout」、値に秒数を指定

2. Java クライアントでの無通信監視タイムアウト時間
・ Java コマンドラインで指定する場合
以下の形式でシステムプロパティを指定
-Djp.co.nec.orb.ClientAutoTimeout=<タイムアウト時間(秒)>
・ J2EE サーバ(Web コンテナ)に指定する場合
Web コンテナ運用管理コンソールの「Java 設定」-「Java VM の引数」に以下の形式でシステムプロパティを追加
-Djp.co.nec.orb.ClientAutoTimeout=<タイムアウト時間(秒)>
・ J2EE サーバ(JNDI サーバ)に指定する場合
JNDI サーバの構成情報ファイルの javaargs(UNIX の場合は USER_JAVAARGS)のキーに以下の形式でシステムプロパティを追加
-Djp.co.nec.orb.ClientAutoTimeout =<タイムアウト時間(秒)>
注）JNDI サーバの構成情報ファイル
Windows　%JNDI_HOME%¥config¥jndiserver.conf
UNIX　/etc/WebOTX/ee/jndi.conf
・ プロセスグループ(EJB コンテナ)に指定する場合
運用管理ツールのプロセスグループを選択し、「設定」-「プロパティ」を選択する。「Java システムプロパティ」タブで、プロパティに「jp.co.nec.orb.ClientAutoTimeout」、値

に秒数を指定

3. オペレーション呼び出しのタイムアウト時間

環境設定ファイル(Windows はレジストリ)に設定

設定名：RequestTimeout

値：＜タイムアウト時間(秒)＞

注）Object Broker 環境設定

・Windows

レジストリエディタを用いて

HKEY_LOCAL_MACHINE¥SOFTWARE¥NEC¥ObjectSpinner¥1

に設定名の文字列エントリを作ります。設定する値はエントリの値として設定します。

数字も文字列として設定してください。

・UNIX

環境変数、環境変数 ORBCONFIG で指定されたファイル、~/.orbconf、/usr/lib/ObjectSpinner/conf/orbconf の順に設定値を検索します。

- 環境変数の場合

setenv OrbRoot /usr/ObjectSpinner

setenv NameServicePort 3400

- 環境変数 ORBCONFIG での指定ファイル

~/.orbconf

/usr/lib/ObjectSpinner/conf/orbconf の場合

OrbRoot=/usr/ObjectSpinner

NameServicePort=3400

4. コネクションラウンドロビンでのオペレーション呼び出しのタイムアウト時間

上記(3)の環境設定ファイル(Windows はレジストリ)に設定

設定名：ConnectionRoundRobinTimeout

値：＜タイムアウト時間(秒)＞

・ サーバ側で設定する場合(AP 応答監視タイマ)

オペレーション実行要求をサーバが受け付けてからその応答を返すまでのタイムアウト値を設定します(既定値：2147483 秒)。このタイムアウト値を設定することで、クライアントが無応答となることを抑止します。このタイムアウト値はサーバでの実行時間のほかに通信に要する時間およびキュー待ち時間を考慮して設定します。各オペレーションの実行時間(予測値)の最大値よりは大きな値を設定する必要があります。設定を反映させるためにはシステムの再起動が必要です。

なお、AP 応答監視タイマを超過した場合、クライアントにエラーは返りますがサーバ AP は処理を継続したままなのでサーバ AP の無応答状態が改善するわけではありません。

サーバ AP の無応答状態を改善させるためには[2.2.4 実行時間タイムアウト]を設定してください。

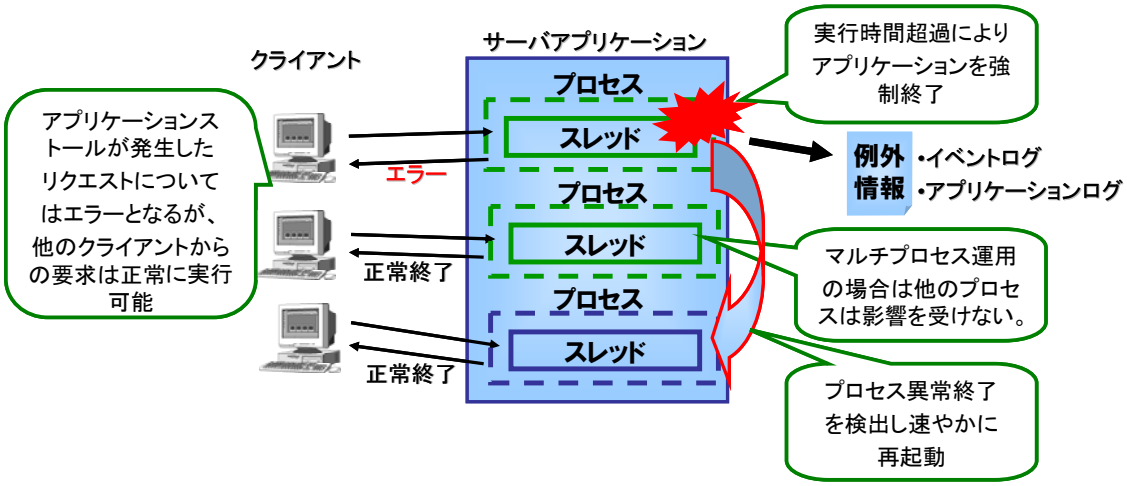
設定方法

運用管理ツールのシステムのプロパティ
[上限設定]-[AP 応答監視タイマ]

2.2.4 実行時間タイムアウト

サーバ側でオペレーション実行を行なってから実行完了までのタイムアウト値を設定します(既定値：上限なし)。この値はサーバでの実行時間のみ考慮して設定します(キュー待ち時間は含みません)。

なお、実行時間タイムアウト値を超過した場合、AP は終了します。AP を再起動させるためには再起動回数（運用管理ツールのシステムのプロパティ[上限設定]-[プロセス障害時の再起動]）を 2 以上にする必要があります。推奨はデフォルトの 50 です。プロセス障害時の再起動回数が 1 で AP が終了した場合、プロセスは終了したままで再起動しませんので注意してください。



設定方法

運用管理ツールのオペレーションのプロパティ
[オペレーションの自動活性]-[実行時間上限]

2.2.5 クライアント無通信監視間隔

サーバとクライアントの間の無通信状態を監視します(既定値：上限なし)。本タイマ値以

上無通信状態(具体的には要求も応答も流れない状態)が続いた場合はコネクションを切断します。設定を反映させるためにはシステムの再起動が必要です。

設定方法

運用管理ツールのシステムのプロパティ
[クライアント制御]-[クライアント無通信監視を行う]-[クライアント無通信監視間隔]

2.3 キューイング数上限

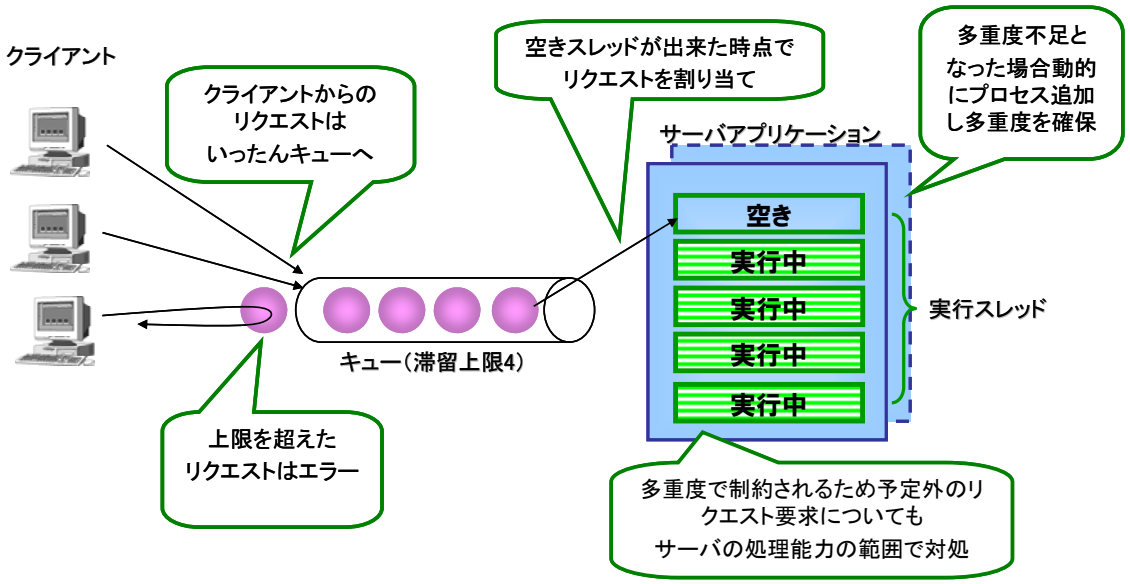
サーバアプリケーションすべてのスレッドがクライアントからのリクエスト処理を行なっている場合、新たな要求はキューイングされ実行待ちとなります。既定値ではメモリの許す限りキューイングされてしまい、クライアントから見るとキュー数が増えれば増えるほど無応答時間が長くなります。ある程度以上のキューイングを抑制することによりクライアントのレスポンスを保証することが出来ます。

(平均待ち時間)=(平均サービス時間)*{(キュー数)+1}

設定方法

運用管理ツールでシステム毎、アプリケーショングループ毎、プロセスグループ毎(後者ほど設定優先度高)に設定できます。

- システムのプロパティ[上限設定]-[キューの最大数]
- アプリケーショングループのプロパティ[キューの最大数]-[キューの最大数]
- プロセスグループのプロパティ[キューの最大数]-[キューの最大数]



2.4 接続クライアント数上限

richクライアントの場合、想定以上の接続を受けるとサーバ側資源(主にメモリとファイルディスクリプタ)を余分に使います。接続数に上限を設けることにより設定以上のクライアントから要求を受け付けなくすることが出来ます。ただし接続クライアント数をぎりぎりに設定してしまうとゴーストセッションが残っている場合、接続できなくなる可能性がありますので、ゴーストセッションの対策をとった上である程度の余裕を持たせてください。ゴーストセッションの対策については「2.2.5 クライアント無通信監視間隔」や「2.7.1 TCP/IPに関する設定について」を参照してください。

設定方法
運用管理ツールのシステムのプロパティ
[上限設定]-[利用可能な同時接続クライアント数]

2.5 1セッションあたりの同時実行数

thin クライアント構成の場合、Web サーバから 1 つのセッションを通して多重にリクエスト要求が発行されます。クライアント数やトランザクション処理件数に応じた 1 セッションあたりの同時実行数を設ける必要があります。

設定方法
運用管理ツールのシステムのプロパティ
[クライアント制御]-[1 プロセス当たりの多重度]

2.6 電文長サイズ

サーバ側の送受信の上限は設定の有無にかかわらず 9,999,998 バイト(10M)となります。また、クライアントが VB または C++の場合、電文の受信最大サイズは 8,388,608 バイト(8M)となります。クライアント側での受信電文長を 10M まで増やしたい場合はクライアントマシンのレジストリ変更が必要です。

設定方法
レジストリの名前：HKEY_LOCAL_MACHINE¥SOFTWARE¥NEC¥ObjectSpinner¥1¥MaxMessageSize
属性：文字列属性(REG_SZ)
値　：0～4294967295(単位:バイト)
意味：受信可能とする最大サイズを指定します。

補足：送信には関係ありません。

2.7 通信に関するチューニング

ブラウザから Web サーバを経由し AP サーバにアクセスする場合の通信に関するチューニングについて説明します。

2.7.1 TCP/IP に関する設定について

全ての通信の基本である OS の TCP/IP に関する設定について説明します。

概要

TCP/IP に関する最も重要な設定は、送信タイムアウト時間と KeepAlive の設定です。例えば、通信相手のサーバマシンの電源が落ちて、TCP/IP の切断処理が行われないままとなった場合、クライアントでは、送信タイムアウト時間と KeepAlive のどちらかの設定によって障害が検出されるまで、通信できない状態になります。

大抵の場合、送信タイムアウト時間の設定によって、長くて 10 分程度で障害を検出することができます。ただ、稀に受信待ちとなった場合には、KeepAlive の OS のデフォルト設定で約 2 時間障害を検出できません。

KeepAlive の設定は、サーバ側で無効なコネクションを破棄するために利用することがより一般的です。システムの障害復旧時間の要件に応じて、これらの設定のチューニングを行ってください。

OS の設定方法

設定内容や設定方法は、次に示す通り、OS 毎に異なります。詳細については、各 OS のリファレンスをご覧ください。

HP-UX の場合）
/etc/rc.config.d/nddconf に次のように設定します。
送信タイムアウト時間（データ送信時）：

```
TRANSPORT_NAME[num]=tcp
NDD_NAME[num]=tcp_ip_abort_interval
NDD_VALUE[num]=600000
```

送信タイムアウト時間（接続要求時）：

```
TRANSPORT_NAME[num]=tcp
NDD_NAME[num]=tcp_ip_abort_cinterval
```

NDD_VALUE[num]=75000

KeepAlive :

次のデフォルトの設定では、tcp_ip_abort_interval の値を加えた２時間１０分で障害を検出します。

TRANSPORT_NAME[num]=tcp

NDD_NAME[num]=tcp_keepalive_interval

NDD_VALUE[num]=7200000

Linux の場合）

/etc/sysctl.conf に次のように設定します。

送信タイムアウト時間（データ送信時）:

次のデフォルトの設定では、13～30 分で障害を検出します。

net.ipv4.tcp_retries = 15

送信タイムアウト時間（接続要求時）:

net.ipv4.tcp_syn_retries = 5

KeepAlive :

次のデフォルトの設定では、7200 秒 + 75 x 9 秒で、約２時間１１分で障害を検出します。

net.ipv4.tcp_keepalive_intvl = 75

net.ipv4.tcp_keepalive_proves = 9

net.ipv4.tcp_keepalive_time = 7200

Solaris の場合）

/etc/system に次のように設定します。

送信タイムアウト時間（データ送信時）:

set tcp:tcp_ip_abort_interval = 480000

送信タイムアウト時間（接続要求時）:

set tcp:tcp_ip_abort_cinterval = 180000

KeepAlive :

次のデフォルトの設定では、tcp_ip_abort_interval を加えた２時間８分で障害を検出します。

set tcp:tcp_ip_keepalive_interval = 7200000

Windows の場合）

HKEY_LOCAL_MACHINE¥System¥CurrentControlSet¥services¥Tcpip¥Parameters キー(レジストリ)の値を次のように設定します。

送信タイムアウト時間（データ送信時）:

次の設定では、アダプタ毎のレジストリ値 TcpInitialRTT の値が 3000 ミリ秒である場合、3 + (3 x 2) + (6 x 2) + (12 x 2) + (24 x 2)秒で、約９３秒で障害を検出します（リトライする度に、リトライの間隔が直前の間隔の２倍になります)。

TcpMaxDataRetransmissions　REG_DWORD　5

送信タイムアウト時間（接続要求時）:

次の設定では、アダプタ毎のレジストリ値 TcpInitialRTT の値が 3000 ミリ秒である場合、3 + (3 x 2) + (6 x 2)秒で、約２１秒で障害を検出します（リトライする度に、リトライの間隔が直前の間隔の２倍になります)。

TcpMaxConnectRetransmissions　REG_DWORD　3

KeepAlive :

次のデフォルトの設定では、7200 秒 + 5x1000 ミリ秒で、約２時間で障害を検出します。

KeepAliveTime　REG_DWORD　7200000

KeepAliveInterval　REG_DWORD　1000

TcpMaxDataRetransmissions　REG_DWORD　5

次のレジストリ値 TcpInitialRTT は、設定場所やデフォルト値が OS 毎に異なります。デフォルト値については OS 毎に確認して頂く必要がありますが、TcpMaxConnectRetransmissions や TcpMaxDataRetransmissions といった再送回数のチューニングだけを行うようにすれば、設定場所まで意識する必要はありません。

TcpInitialRTT REG_DWORD　3000

WebOTX で KeepAlive を有効にするための設定方法

運用管理ツールのシステムのプロパティ

[クライアント制御]-[TCP レベルでのアライブチェックを行う]

Oracle クライアントで KeepAlive を有効にするための設定

WebOTX で利用することが多い Oracle クライアントでの設定方法について説明します。詳細については、Oracle のリファレンスを参照してください。

・Oracle thin ドライバで KeepAlive を有効にするための設定

JDBC データソースのデータソース名[dataSourceName]に設定する接続文字列として、次の形式（例）で設定してください。

"jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=xxx.xxx.xxx.xxx)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=ORCL))(ENABLE=BROKEN))"

・Oracle OCI ドライバおよび C++アプリケーションで KeepAlive を有効にするための設定

Oracle Net Service の tnsnames.ora ファイルに、次の形式（例）で設定してください。

```
ORCL=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=xxx.xxx.xxx.xxx)
(PORT=1521))
(CONNECT_DATA= (SERVICE_NAME=ORCL))(ENABLE=BROKEN))
```