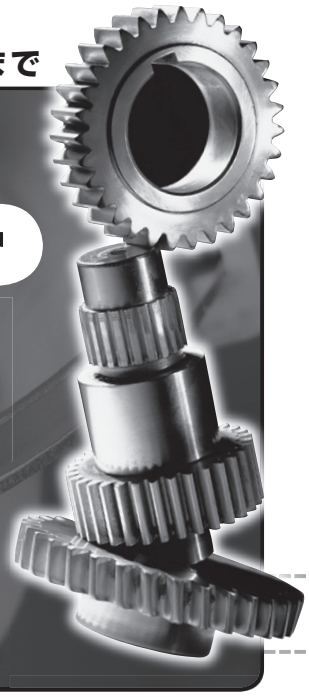
特集3 プロセス／スレッドから メモリ管理、クラスタリングまで

Windows + SQL Server 安定稼動のベストプラクティス



Special Feature 3

SQL Serverを安定稼動させるためには、さまざまな設定や運用管理が必要となると同時に、SQL Serverを動かしているOSとしてのWindowsについて詳しく知っておくことも重要である。本特集では、まずSQL Serverが使用しているWindowsの機能概要を紹介したうえで、SQL Serverを安定稼動させるためのテスト／設定／運用における注意点について紹介する。WindowsとSQL Serverの相関関係をしっかり押さえ、効率的な運用管理を行なうための指針としてほしい。

ユニアデックス株式会社
内ヶ島暢之 UCHIGASHIMA, Nobuyuki

Windowsのアーキテクチャを理解する

Windowsの機能と役割

ご存知のように、Windowsはハードウェア上でソフトウェアを動かすための基本ソフトウェア（OS）であり、テクノロジーの進化と共にさまざまな機能を実装してきた。アプリケーションは必ずWindows上で動作し、プロセッサでの計算やディスク／画面といったハードウェアデバイスへの入出力を行なうが、OSはアプリケーションが直接ハードウェアにアクセスすることをガードしている。そのためWindowsはハードウェアへのアクセスを抽象化し、アプリケーションが必要とする機能（例えばファイルの入出力）をAPI（Application Programming Interface：プログラムから呼び出しできる関数）として公開している。また、Windowsは複数のアプリケーションを同時に動かすためにリソース配分の調整なども制御している。

Windowsとアプリケーションの概念図を**図1**に示す。Windowsには「ユーザーモード」と「カーネルモード」の2つの実行モードがあり、一般的にSQL Serverのようなアプリケーションや Windowsの機能の一部はユーザーモードで動作する。一方で、Windowsの核となる機能やデバイスドライバなどはカーネルモードで動作する。Windowsはハードウェアのメモリ保護機能を利用してユーザーモードからカーネルモードへの直接アクセスを禁止し、アプリケーションによるWindowsの破壊を防いでいる。

アプリケーションが利用できるAPIはkernel32.dllやuser32.dllといった動的リンクライブラリで提供され、必要に応じてAPIを呼び出すことができる。ただし、kernel32.dllなどは単に呼び出しのためのインターフェイスであり、実際はユーザーモードで動作するntdll.dllやカーネルモードで動作するntoskernel.exe[注1]、win32k.sys[注2]の非公開関数として実装されている。呼び出されたAPIは適宜OSの機能に振り分けられ、各機能を多層的に経由してデバイスドライバやハードウェア抽象化層（HAL）[注3]を通って実デバイスへ処理を依頼する。Windowsの代表的な機能にはプロセス制御、メモリ管理、ファイルシステムキャッシュ、I/Oマネージャ、構成マネージャ（レジストリ）などがある。HALは、さまざまなハードウェア仕様の相違をWindowsから見て同じインターフェイスで制御可能にするための機構なのである。

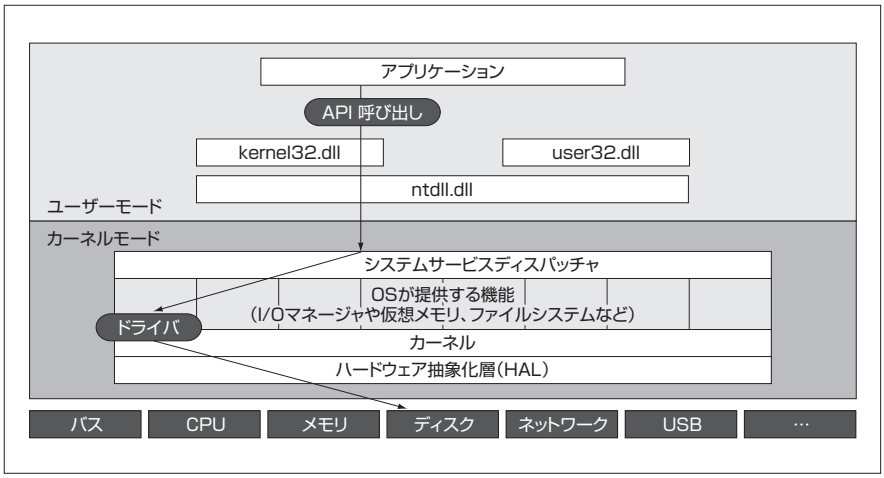
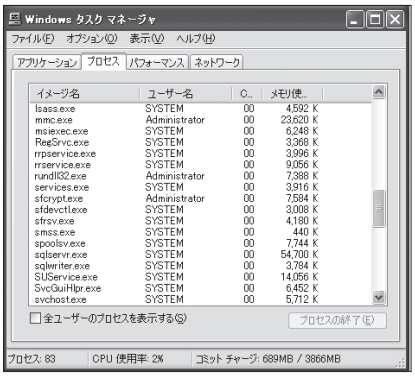


図1：Windowsとアプリケーションの関係概略図

プロセス／スレッドとスケジューリング

プログラムを実行する際、Windowsは最初にプロセスを作る。プロセスはプログラムを実行するために必要な情報を格納しているオブジェクトである。Windowsのタスクマネージャを確認すると、プロセスのイメージ名にプログラムの名前

注1：OSのカーネルとシステムサービスを提供している実体。
注2：ウィンドウ管理やグラフィック操作関数を提供するドライバ。
注3：カーネルやドライバからの操作のためにマザーボードの違いなどを吸収するコード群。



画面1：タスクマネージャにおけるプロセス一覧。イメージ名列にプログラムの実行ファイルの名前が入っている

が表示されるためプロセスとプログラムは同じもののように見えるが、根源的には違うものである（**画面1**）。プログラムは命令の集合体であり、プロセスはプログラム実行時に使用される各種リソースの集合である。プロセスには必ずスレッドが1つ以上存在するが、スレッドとはOSから見たときの実行最小単位であり、CPUの割り当てを行なう単位でもある。

では、Windowsはスレッドに対しどのようなアルゴリズムに基づいてCPUの割り当てを行なうのだろうか。スレッドが取り得る状態の中で重要なものを**表1**に示す。Windowsのバージョンや種類によって取り得る状態はいくつものバリエーションが存在する。

次に、現在のWindowsが「プリエンプティブ」なスケジューリングを行なっているということを知っておく必要がある。プリエンプティブとは、プログラムがCPUを使用しているとき、WindowsがCPUを必要とする際にはその実行を強制的に「横取り（プリエンプション）」できることを言う。ユーザープログラムに問題が発生しCPUを独占し続けたとしてもWindowsが実行権限を強制的に横取りするため、他のプログラムやWindowsへの影響を軽減できる。一方で、そのようなスケジューリングはWindows側の実装が複雑になる（**図2**）。

スケジューリングには、さらに優先度とタイムスライスの概念がある。Windowsは複数のスレッドが実行可能な状態のとき、優先度の高いもの（ハードウェアの割り込みやタイマー処理など）から割り当てを行なう。またタイムスライスとはスレッドが実行できる時間の最大値であり、タイムスライスで規定された時間まで実行を続けたスレッドは実行権限を他のスレッドに明け渡される。

表1：スレッドの状態と説明。ほかにも多数の状態が存在する

状態	説　明
レディ	CPUの実行キュー上に置かれており、CPU割り当てが行なわれるのを待っている
実行	CPUが割り当てられ実行されている。実行が終わるとレディや待機に移行する
待機	実行同期やI/O完了待ちなどの条件によって待機している。待機が終了するとレディに移行する

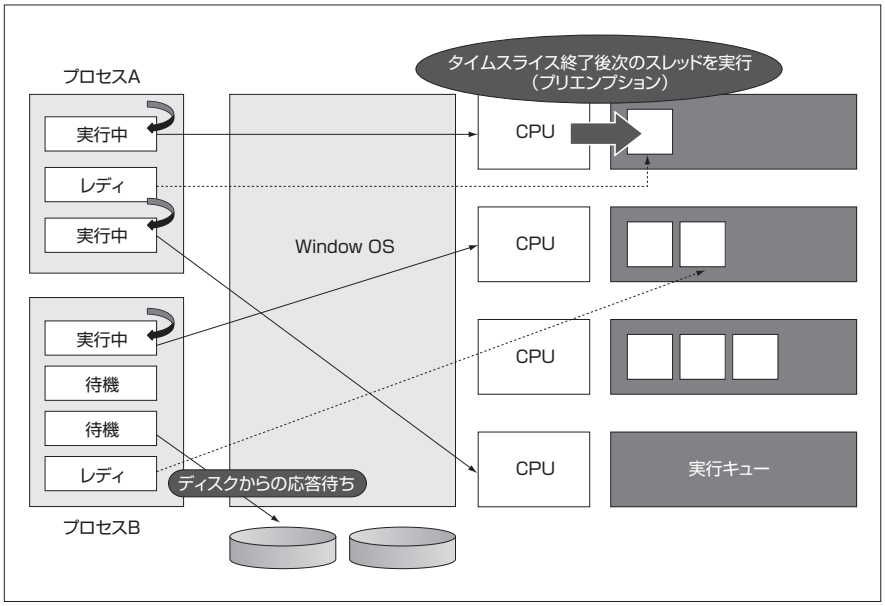


図2：プロセス、スレッド、CPUの関係概略図

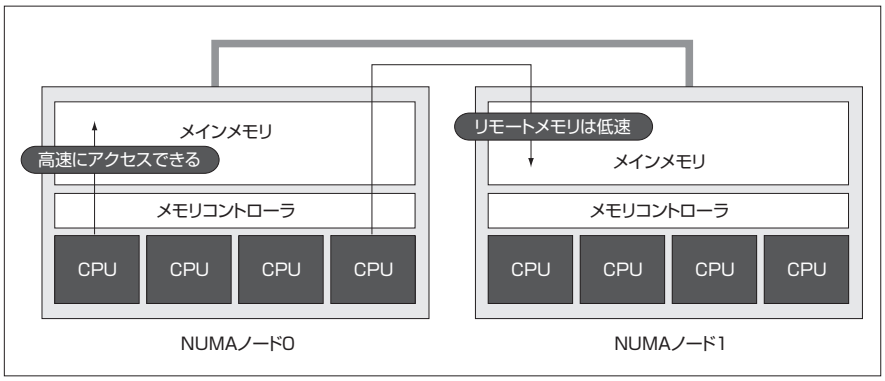


図3：NUMA概略図。NUMAノード間の接続方式はハードウェアベンダによって異なる

エンタープライズ領域でのサーバー機では、利用できるCPU数の増加とマルチコア化が進んでいる。CPUを単純に増やしたとしても、CPUとメモリを結ぶ通信回路（バス）がボトルネックとなり、メモリアクセスが遅くなるため処理効率改善とならないケースがある。CPUスケーラビリティ向上のため「NUMA（Non-Uniform Memory Access）」と呼ばれるアーキテクチャが導入され、Windows Server 2003以降で対応した。NUMAはCPUとメモリをグループ（NUMAノード）化し、グループをまたがったメモリアクセスを減らしてメモリアクセスの効率向上を目的とする。通常NUMAノード間は高速なバスで接続されているが、ローカルメモリへのアクセスのほうがリモートメモリへのアクセスよりも高速である。NUMAのメリットを享受するには、アプリケーションがNUMAであることを意識した構成にする必要がある。SQL Serverは2005以降でNUMAに対応している。NUMAの概略図を**図3**に示す。

メモリ管理

アプリケーションはプログラムの実行イメージ（命令語）やデータを「メモリ」上に配置する。アプリケーションから利用できるメモリは「仮想メモ

リ」と呼ばれ、ハードウェアとしてマザーボード上に存在する「物理メモリ」とは区別される。

仮想メモリはプロセスに対して割り当てられる仮想的なメモリ空間であり、VirtualAlloc[注4]などのAPIから利用できる。一方、OSとCPUは各プロセスが持っている仮想メモリを物理メモリに変換してアプリケーションからのメモリ要求に応えている。メモリ変換機構は各プロセスに対してプライベートなメモリ空間を提供し、アプリケーションは実装されている物理メモリサイズ以上の空間を利用できる。また、プライベートなメモリ空間であることから、他のプロセスからのアクセスを防ぐこともできる。Windowsのプロセス用仮想メモリ空間の最大値を**表2**に示す。

例えば、物理メモリが1GBしかない32ビット環境で仮想メモリ空間いっぱいの2GBを利用した場合、残りの1GBとほかに動いているプログラムやOSが使うメモリ空間はどこにいくのだろうか。実は、仮想メモリと物理メモリの変換機構が解決している。OSでは仮想メモリの管理をページと呼ばれる単位で行ない、アプリケーションなどに対するメモリ割り当てを物理メモリに行なう。物理メモリが足りなくなったときは頻繁に使われていないページをページファイルと呼ばれるディスクファイルに待避する。これを「ページアウト」と呼ぶ。待避したページを再び物理メモリ上にロードする処理を「ページイン」と呼び、ページアウトと併せて「ページング」と呼ぶこともある。ページングは実メモリ以上の領域をプロセスが使うことを許容するメリットがあるが、メモリアクセスにディスクアクセスが加わるためパフォーマンス劣化の原因となることがある。**図4**に仮想メモリの概要を示す。

表2：各アーキテクチャの仮想メモリ空間最大値

CPUアーキテクチャ	仮想メモリ最大値	特　徴
x86（32bit）	2GByte	32ビットは最大4GBを表現でき、WindowsではOSに2GB、ユーザーに2GBを割り当てる。OSの設定によってユーザー空間を3GBに拡張可能
x64（64bit）	8TByte	64ビットでは論理的には2の64乗で16エクサバイトだが、現在のWindowsの仕様上の制限が存在する
IA64（64bit）	7TByte	

表3：代表的なRAID構成。実効容量とは最低本数に対して何本分のデータを格納できるかを示している。100GBのディスク装置3本でRAID5を構成すると、実質200GBの書き込みが可能

RAID	耐障害性	最低本数（実効容量）	特　徴
0	なし	2（2）	ストライピング。複数のディスク装置に読み込みと書き込みを分散させる
1	あり	2（1）	ミラー。複数（一対）のディスク装置に同じ内容を保持する
5	あり	3（2）	パリティ付きストライピング。1台のディスク装置に障害が発生してもパリティ計算することで失われたディスクのデータを再現できる
10	あり	4（2）	RAID0と1の組み合わせ。実効容量が小さくなり相対的にコストが大きくなる

表4：NTFSで実装されている機能の一部。ほかにも多数の機能が実装されている

機能	機能概要
データ回復機能	NTFS以前のファイルシステムではファイル入出力中にシステム障害が発生するとボリュームが破損することがあった。データベースのように書き込みログを保持することでボリュームの破損を最小限にとどめることができる。保護対象はボリュームであり、ファイルではない。論理的な書き込み保証はアプリケーションで考慮する必要がある
セキュリティ	Windowsセキュリティモデルで実装されているアクセス記述子やアクセスコントロールを用いてディレクトリやファイルレベルでユーザーのアクセス制御が可能。セキュリティモデルはOS内でのオブジェクト制御の一部なのでプロセスやスレッドなどと同じ機構でセキュリティ機能が実装されている。セキュリティ機能をさらに利用し、ユーザーごとのボリューム利用制限（クォータ）が可能
圧縮	ファイル、ディレクトリ、ボリュームのレベルでそれぞれ圧縮が可能。圧縮と伸張はユーザーアプリケーションに対して透過的に行なわれるため、Read/Write要求時に圧縮であることを意識しなくても良い（非圧縮時と同じAPIでアクセスできる）
スパースファイル	ファイル中で利用されていない領域をディスク上で割り当てない機能。例えば、見た目が100MBのファイル内で利用しているのが20MBだとすると、スパースファイルであれば実ディスク上に確保される領域は20MBとなる
暗号化（EFS）	EFSは「Encrypting File System」の略称。ユーザーアカウントに紐付く公開鍵と秘密鍵を利用してファイルの暗号化を行なう。圧縮機能と同様にユーザーアプリケーションに対し透過的に暗号化される。圧縮と同時には使用できない。なおWindows 2008から実装されているBitLockerも暗号化機能だが、EFSとは別のもの

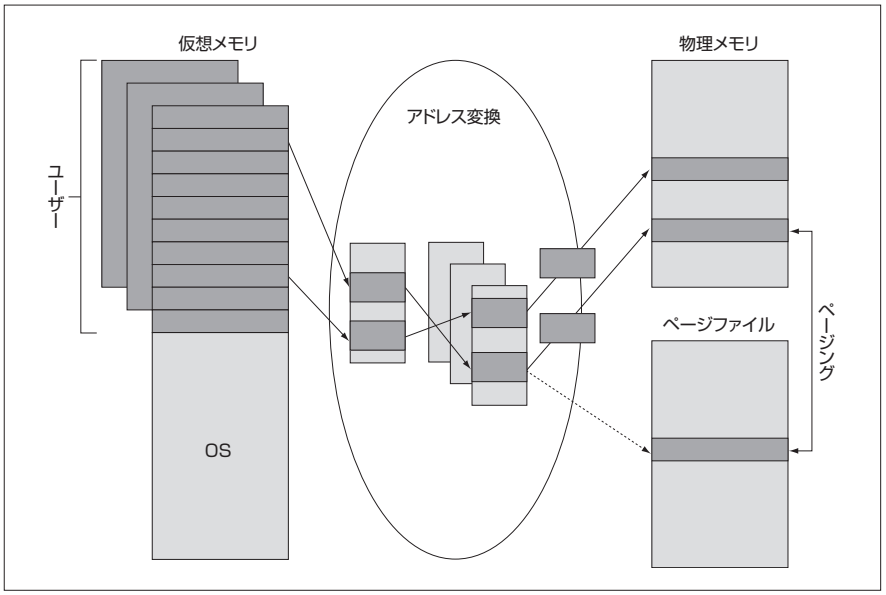


図4：仮想メモリと物理メモリの関係。OS内部では仮想アドレスをいくつかに分割し、カーネル内部で保持しているページディレクトリ、ページテーブルという構造体を使って物理メモリアドレスに変換する

ファイルI/O

ファイル入出力を行なう場合、アプリケーションはファイルに書き込みを行なう。一方でOSはファイルシステムを1つまたは複数のディスク装置上に作成し、ファイルへの入出力の要求に応答している。エンタープライズ環境ではディスク装置は何かしらのRAID（Redundant Arrays of Independent Disks）構成が組まれていることが多い。RAIDは複数のディスクを組み合わせることで耐障害性や負荷分散を行なう重要な概念である。RAIDはWindowsの機能（ソフトウェアRAID）でもハードウェアの機能（ハードウェアRAID）でも構成できるが、ハードウェア機能の利用が一般的である。RAIDの種類を**表3**に示す。

注4：仮想メモリを割り当てるためのAPI。ほかにもVirtualFreeやVirtualLockなどがある（http://msdn.microsoft.com/ja-jp/library/cc430204.aspx）。

Windowsは、構成されたRAIDディスクの上にファイルシステムを作成することで、ボリュームやファイル、ディレクトリを扱うことができる。Windowsではさまざまなファイルシステムをサポートしているが、現在ではデータベースのデータファイルを配置するような場合はNTFS（NT File System）が利用されることが多い。NTFSはエンタープライズ領域での利用を想定して設計されており、ファイルシステムドライバとしてさまざまな機能が実装されている。その一部を**表4**に記載する。

ネットワーク

Windowsは設計当初（今ほどネットワーク通信が一般的ではなかった）からネットワーク通信を意識した設計が行なわれ、多数のネットワーク機能が提供されている。アプリケーションにはネットワークAPIとして機能が提供され、内部ではOSI参照モデル[注5]の各層にWindowsコンポーネントが対応した実装をしている。**表5**にOSI参照モデルとWindowsコンポーネントの対応を示す。ここでは、通信の仕組みとしてネットワークAPIで提供されている「Winsock」と「名前付きパイプ」を紹介する。

WinsockはWindowsに実装されているソケット通信の仕組みである。通信サーバーはソケットと呼ばれる通信エンドポイントを作成し、ソケットとサーバーのアドレスとポート番号を紐付ける。クライアントもソケットを作成し、対象サーバーのアドレスとポートを指定し接続する。アドレスだけではなく、ポートを指定することでクライアントは特定のサーバーアプリケーションに接続できる。接続後、クライアントとサーバーはソケットを通してデータ送受信を行なう。

名前付きパイプは¥¥server名¥pipe¥pipe名の形式で提供される通信端点である。アプリケーションはパイプ名を指定することでサーバーと通信できる。「server名」にはDNS名やIPアドレスを指定でき、「pipe」は固定名、「pipe名」はサブディレクトリを含む固有の名前を記述できる。**図5**にWinsockと名前付きパイプでの通信概要図を示す。

クラスタリング

Windowsでは、クラスタリング機能として「ネットワーク負荷分散クラスタ」と「フェイルオーバークラスタ」の2つを提供している。ここではフェイルオーバークラスタについて取り上げよう。

フェイルオーバークラスタは、Windows Server 2003までは「Microsoft Cluster Service（MSCS）」、Windows Server 2008では「Microsoft Failover Cluster（MSFC）」と呼ばれていた耐障害性を高める技術である。フェイルオーバークラスタでは複数のサーバーでアクティブ／パッシブ構成を取り、アクティブ側のサービ

表5：OSI参照モデルとWindowsコンポーネントの対応。プレゼンテーション層以下はOSやドライバで実装されている。NDISとはマイクロソフト社と3Com社が決めたネットワークインターフェイス仕様

OSI参照モデル	Windowsコンポーネント		説明
アプリケーション	ネットワークアプリケーション		ユーザーモードで動作するアプリケーション。ネットワークAPIをコールする
プレゼンテーション	ネットワークAPI DLL		プロトコルに依存せずにアプリケーションがネットワークを利用するための実装
セッション		ネットワークAPIドライバ	
トランスポート	プロトコルドライバ（TCP/IP、NetBEUI、IPX/SPXなど）		Windowsで提供されるネットワークプロトコルの実装
ネットワーク			
データリンク	NDISライブラリ	NDISミニポート	NDISの実装
物理	Ethernet、IrDAなど		物理的な通信手段の提供

表6：性能テストにおける注意点。本番稼動後と同じ状況をシミュレートすることが重要である

注意すべき点	怠った場合の稼動後リスク
本番を想定したセッション数／処理数をシミュレートする	高負荷時にのみ発生する問題が露呈する
本番同等のデータ内容／データ量をテストデータとして用意する	クエリがテスト時と違い、遅いプランが選択される
テストを可能な限りリリースする本番環境で行なう	本番ハードウェアの差異による性能差を確認できない
クエリのレスポンスだけでなくサーバー稼動状況も確認する	リソース逼迫を確認できない

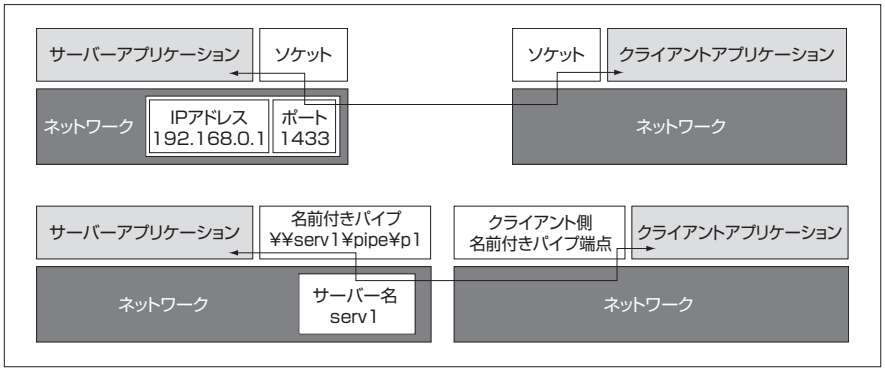


図5：Winsockと名前付きパイプの通信概略図。それぞれアプリケーションはソケットやパイプに対しデータを送り込むことで通信を行なっている

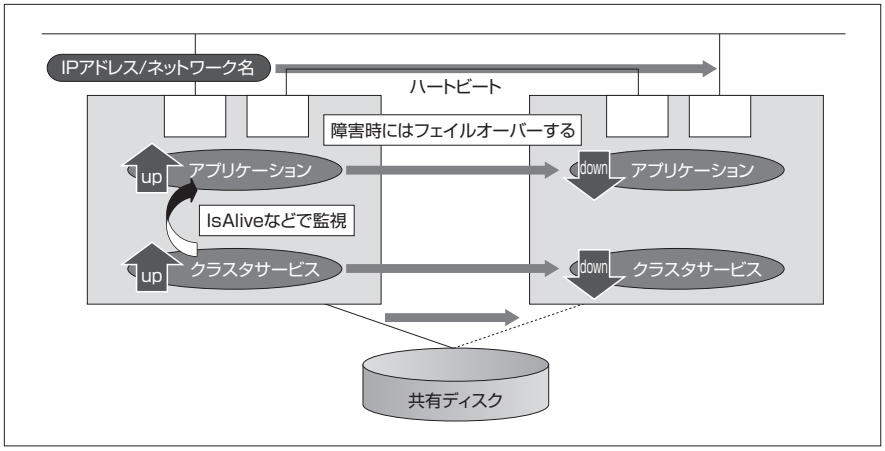


図6：2ノード構成フェイルオーバークラスタの概略図。実際には2つのサーバーがドメインに参加している必要がある。共有ディスクに作成されたファイルシステムもクラスタに管理され障害時にフェイルオーバーする

注5：国際標準化機構が策定した、異機種間コンピュータ通信のためのモデル。7つの階層から構成されている。

スに異常が発生した場合にパッシブ側に自動的にサービスを移動(フェイルオーバー)し、継続できる。MSFCを利用すれば障害発生からサービス再開を自動化でき、停止時間を短くできる。

クラスタサービスはサーバー間で「ハートビート」と呼ばれるネットワーク通信を使って定期的に互いの死活確認を行なっており、死活確認が途切れ、異常と判断されたサーバーをクラスタメンバーから切り離す。また、サーバー間だけでなくサーバー内ではクラスタリソースの死活確認を行なっている。管理できるクラスタリソースはクラスタサービスそのものやネットワーク、サーバー名、ディスク、SQL ServerやIIS(Windows標準のWebサーバー)のようなアプリケーションも含まれる。

クラスタサービスは「IsAlive」と「LooksAlive」と呼ばれる2種類のリソース監視を行なっている。IsAliveは詳細確認、LooksAliveは概要確認である。リソース監視は「リソースDLL」と呼ばれるDLLとして実装され、それぞれのリソースタイプごとに存在する。クラスタサービスから各クラスタリソースに対するIsAliveとLooksAliveのポーリング間隔を決めることができる。

フェイルオーバークラスタを構成するためには、ドメイン環境、ネットワーク、共有ディスクが必須である。フェイルオーバークラスタの概略図を**図6**に示す。

Windows+SQL Server活用のベストプラクティス

本番稼動前に行なうべきテスト

SQL Serverには多数の設定項目がある。インストール直後は一般的な設定がすでになされている項目が多いが、必ずしもその設定がシステムに最適とは限らない。なぜならば、最適な設定は十分なテストや稼動状況を確認したうえでようやく決まるものだからである。

SQL Serverをインストールし、システムをリリースする前には必ず性能テストや運用テストを実施すべきである。なぜなら、稼動後に問題が発覚した場合、その多くは設定変更が及ぼすリスクを想定するのが難しいためである。システムの潜在的な問題はテストを通じて可能な限り洗い出しておく必要がある。一般的なSQL Serverのインストールから本番稼動までの流れを**図7**に示す。また、性能テストを行なううえで気をつけるべき点を**表6**に示す。

設定項目

表7に各OSカテゴリにおけるWindowsやSQL Serverの設定項目、設定値の決め方、注意点を挙げる。当然のことながらシステムごとに個別に注意すべき点はほかにもあるが、ここでは最初に目を向けるべき設定を挙げた。性能テストの結果から項目変更が必要か検討してほしい。

表7:設定を検討する、または注意すべき項目

カテゴリ	設定項目	設定指針
CPU	max degree of parallelism	クエリ並行実行の最大数を決める。性能テストの結果、CPUリソースの高騰や並行実行スレッド間のロック待ちが長い場合は制限する。クエリにmaxdopヒントを付与して制御することもできる
	Affinity Mask	SQL Serverが使うCPUを決める。サーバーでSQL Server以外のアプリケーションにCPUリソースを割り当てたい場合に設定する
メモリ	max server memory	SQL Serverが利用するメモリサイズを設定する。既定では自動管理のため、他のアプリケーションで利用できる物理メモリが不足することがある。(物理メモリ)−(OSや他のアプリケーションが使うメモリ量)を設定すると良い
	Lock Pages in Memory権限	SQL Serverが確保するメモリをなるべくページアウトさせないように起動アカウントに付与する権限(OSの設定)。max server memoryと同時に利用し必要な分だけ物理メモリを確保できる
I/O	自動拡張設定	最善の方法は自動拡張せず、必要な領域を手動で割り当てるようにすることである。自動拡張をする場合は「比率」では設定せずサイズ指定する。100GBの10%は10GBであり10GBの拡張が終わるまでデータ操作がブロックされてしまう
	tempdbのファイル配置	tempdbはソートやスナップショットのバージョンなどで必要な一時領域が格納されるため、データ領域とは異なるドライブに配置する。またコア数分だけ同じ大きさのファイルの数を配置することで領域確保の競合を回避できる
ネットワーク	リモートDAC(Dedicated Admin Connection)	SQL Serverが接続応答を返さないときなど、トラブルシュート用に利用できる接続設定。既定ではネットワークアクセスが「Off」になっている
	ファイアウォール	TCPや名前付きパイプで接続する際に接続を阻害しないようにファイアウォールの設定は必要なポート解放または例外設定をしておく
クラスタ	IsAliveの設定	クラスタ監視処理の試行間隔を調整できる。高負荷時には一時的にIsAlive監視が失敗しフェイルオーバーを起こすことがある。高負荷時に誤検知しないように試行間隔を延ばすことができる
	Priority Boost	SQL Serverのプロセス優先度を上げる設定。本設定はOnにしないこと。Onにするとクラスタ監視動作が阻害され、不必要なフェイルオーバーが発生する。設定をOnにしても大きな効率向上はしない。既定はOffである

定常運用時に採取する情報

十分な性能/運用テストを終えてリリースされたシステムであっても、本番稼動後に何かしらの問題が発生するものである。このような場合、何も設定を変えていなくても障害発生時に何が起きたのか知ることができる場合もあるが、パフォーマンスの問題に関して言えば後から原因を追究する資料として利用できるような情報は既定では採取されていない。SQL Serverのパフォーマンス値を採取するにはサードパーティ製のツールを含めさまざまあるが、ここではOSのパフォーマンスカウンタで通常運用時に最低限採取しておくべき情報を**表8**にまとめた。

これらのパフォーマンスカウンタの設定は、性能テスト時に同じ設定で採取しておき、採取による負荷が問題にならないこととパフォーマンス値に問題がないことの双方を確認しよう。採取イン

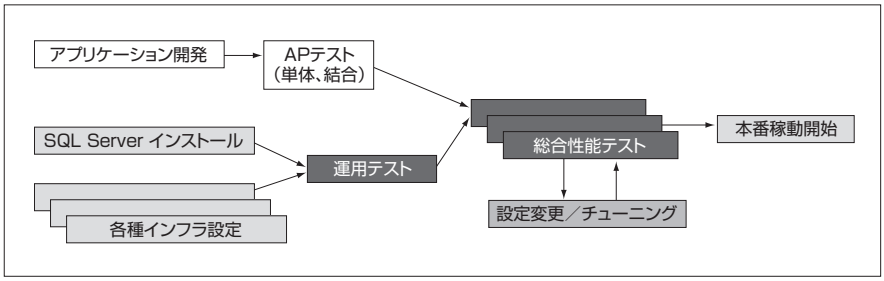


図7:SQL Serverが関連する開発作業の主な流れ。総合性能テストは複数回実行され、都度効率改善のための施策がなされる。運用テストではミドルウェアの機能テストやバックアップテストなどが行なわれる。稼動前に性能を確認することでSQL ServerやWindowsの設定を既定のものからどのように変更すれば良いかが判断できる

表8：採取すべきパフォーマンスカウンター覧。本項目はWindows 2008 R2とSQL Server 2008がインストールされている環境から抜粋、記載した。各項目の詳細はMSDNやパフォーマンスカウンタの説明を参照

SQL Serverのカウンタ		OSのカウンタ	
SQL Server：Buffer Manager	Buffer cache hit ratio	Memory	Available Bytes
	Page life expectancy		Page Faults/sec
SQL Server：Databases	DataFile(s) size(KB)		Page Reads/sec
	Transactions/sec		Page Writes/sec
SQL Server：General Statistics	Logins/sec		Pool Nonpaged Bytes
	Logouts/sec		Pool Paged Bytes
	User connections	PhysicalDisk	Avg. Disk sec/Read
SQL Server：Latches	Average Latch Wait Time(ms)		Avg. Disk sec/Write
	Latch Waits/sec		Current Disk Queue Length
	Total Latch Wait Time(ms)		Disk Read Bytes/sec
SQL Server：Locks	Average Wait Time(ms)		Disk Write Bytes/sec
	Lock Requests/sec	Process	% Privileged Time
	Lock Wait Time(ms)		% Processor Time
	Lock Waits/sec		% User Time
	Number of Deadlocks/sec		Handle Count
SQL Server：Memory Manager	Target Server Memory(KB)		Thread Count
	Total Server Memory(KB)		Working Set
SQL Server：SQL Statistics	Batch Requests/sec	Processor	% Privileged Time
	SQL Compilation/sec		% Processor Time
	SQL Re-Compilation/sec		% User Time
SQL Server：Transactions	Free Space in tempdb(KB)		Working Set
SQL Server：Wait Statistics	Lock waits		
	Log buffer waits		
	Log write waits		
	Network IO waits		
	Non-Page latch waits		
	Page IO latch waits		
	Page latch waits		
	Wait for the worker		

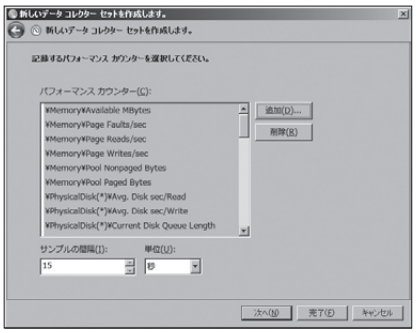
LIST1：ブロッキング確認クエリとその出力例。本例ではsession_id 55のクエリが53を待ち、53のクエリが52を待っている

```
select blocking_session_id, wait_duration_ms, session_id
from sys.dm_os_waiting_tasks
where blocking_session_id is not null

blocking_session_id    wait_duration_ms   session_id
-------------------    ----------------   ----------
53                     33624              55
52                     75528              53
```

表9：リカバリに関する考慮事項。バックアップリストアには共有ディスクやバックアップソフトウェアの機能を利用する場合もある

検討項目	説　明
復旧モデル	リカバリ要件により選択する。単純復旧モデルにすればログバックアップが不要となるが、バックアップ時点にしか復旧できない
フルバックアップの頻度	1日1回のフルバックアップが採取できていると良い。頻度が低い場合は復旧に時間を要することとなる
ログファイルのバックアップ頻度	復旧モデルが単純復旧モデルではない場合は1日内に複数回実行すると良い。頻度が低い場合は復旧に時間を要することとなる
システムデータベースのバックアップ	masterにはデータベース情報、ログイン、構成値などが、msdbには警告やジョブの設定が含まれるためバックアップに含める必要がある
リカバリ方法の手順化	手順書作成とスクリプトや運用ジョブを作成し、手順を標準化する。可能であれば定期的に別サーバーでのリカバリリハーサルを行なう
リカバリ時間測定	テストの中でバックアップメディアからのリストアから業務復旧まで要する時間を測定し、SLAを遵守できるか確認する
dbcc checkdb運用	データベースの整合性を保証するために定期的にdbcc checkdbコマンドを実行し、データベースの破損を早めに検知／対処できるようにする



画面2：パフォーマンスカウンタ設定画面の例。「サンプルの間隔」で取得間隔を調整する。画面はWindows 2008 R2とSQL Server 2008の環境で設定した

スタンスは可能であればすべてのインスタンスを対象とする。パフォーマンスログの設定には取得間隔を設定できるが、間隔が短い場合はファイルサイズが大きく負荷が高くなるが細かい粒度で現象を確認できる。長い場合はファイルサイズが小さく負荷は低いが粒度が荒くなる。現象の発生時間が短い場合には、採取間隔が長いと事象を捉えられない可能性があるため注意すること。取得間隔の設定画面を**画面2**に示す。

DMVの利用

SQL Server 2005以降では、DMV（動的管理ビュー）が実装されている。DMVはSQL Server内部の状況をビューや関数の形式で表示できる機能である。DMVやシステムテーブルを参照する監視／調査スクリプトを事前に作成しておくことで、問題発生時に迅速に対応できる。例として、ここではロックによるブロッキング発生時の確認方法を紹介しよう。**LIST1**にブロッキングを確認するクエリとその結果例を示す。このようなクエリは問題発生時にDAC[注6]から実行でき

注6：SQL Serverがハングしても、問題解析のために利用できる特別な接続方法。

るよう準備しておくのも良い。

データベースのメンテナンス

データベースが本番稼動を開始した後、データベース管理者は何もしなくて良いわけではない。SQL Serverはメンテナンスの手間が少なくなるように動作するが、決してメンテナンスフリーにはならないのだ。ここでは、運用時に考慮しなければならないことをいくつか紹介する。

バックアップリカバリとデータ整合性

データベースのバックアップはデータ保護のための必須手段である。最近はハードウェアの信頼性が高まり、データすべてが失われるような障害は少なくなってきてはいるが、ユーザーが誤ってデータを削除したケースやハードウェア障害でリカバリ作業が必要となるケースはまだ数多く存在する。**表9**にリカバリに関する考慮事項をまとめた。

統計情報の更新

SQL文を効率的に実行するために、クエリオプティマイザが結合方法やアクセス方法を最適化し、実行プランを作成する。最適化のためのインプットとなるものが統計情報である。統計情報は列のデータ分布密度やヒストグラム、列の平均データ長などを含んでいる。そのため統計情報が実際のデータ分布と乖離している場合、クエリオプティマイザは正しく最適化を行なうことができず、SQL文の実行が遅くなる可能性がある。

統計情報は既定では自動的に作成／更新される。しかし、統計情報を作成する際にすべてのデータを使用するわけではない。既定では自動でサンプル率を決定し作成する。また、データの更新が発生するたびに行なわれるわけではないため、必ずしも最新の情報は維持していない。

以上のことから、既定の動作通り統計情報の自動更新はONに設定する。また、運用上大きくデータが入れ替わる（例えばバッチ処理実行やデータロードが行なわれる）タイミングや統計情報が古いことでアプリケーションが効率的なプランを選択していない場合は可能な限りFULL SCANオプションを使い、運用時間が制限されている場合にはできるだけ高いサンプリング率を指定して最新かつ正確な統計情報を維持することを推奨する（**図8**）。

LIST2：断片化確認の例。tb1表が30%以上の断片化を示しているため、再構築を行なうべきである

```
SELECT OB.name,PS.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats
(DB_ID(N'testdb'),OBJECT_ID(N'dbo.tb1'),NULL,NULL,'DETAILED') AS PS,sys.objects AS OB
where PS.object_id=OB.object_id;

name    avg_fragmentation_in_percent
------ ----------------------------
tb1      42.8571428571429
```

LIST3：AdvenrureWorksLTで実行したサンプルクエリ。CustomerIDが30019の顧客の氏名と製品オーダーの支払額を表示する

```
select c.FirstName,c.LastName,SUM(TotalDue) as Total
from SalesLT.Customer c join SalesLT.SalesOrderHeader o
on c.CustomerID=o.CustomerID
where c.CustomerID=30019
group by c.FirstName,c.LastName
```

表10：インデックスのメンテナンス方法と指針。再構築の場合、Enterprise Edition、Evaluation Edition、Developer Editionではオンライン再構築が可能。それ以外のエディションでは選択できず、再構築中はインデックスが利用できない

断片化度合い	方法	利用する実行クエリ	オンライン実行
5～20%程度	インデックスの再構成	alter index reorganize句	可能
30%以上	インデックスの再構築	alter index rebuild句	Editionにより可能

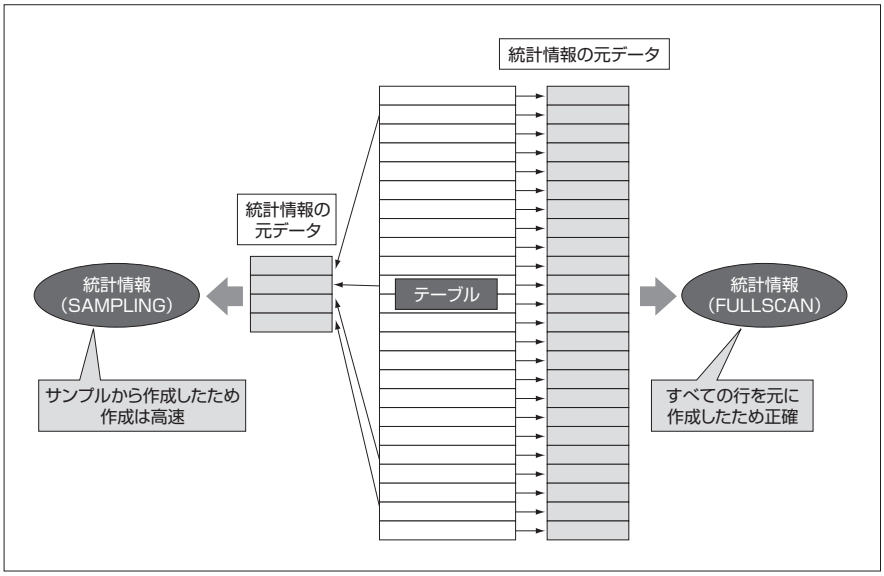


図8：統計情報の作成。作成対象となる元データが多いほうがより正確な統計情報を作成できる。ただし、サンプリングしたほうが作成は高速

インデックス断片化の解消

SQL Serverを長期間利用していると、データの更新に基づいてインデックスが断片化していく。断片化が顕著になるとページやエクステントのスイッチによりデータアクセスに時間がかかるようになり、結果としてクエリの実行時間が長くなる。そのため、インデックス断片化をモニタリングし、断片化を解消する必要がある。モニタリングにはDMF（動的管理関数。DMVと同じようにSQL Serverの内部を知る手段の1つ）のsys.dm_db_index_physical_statsを用いる。実行例を**LIST2**に示した。avg_fragmentation_in_percent列から断片化度合いを確認できる。また、断片化の解消には2つの方法がある。**表10**にその方法を示す。

クエリプランの確認

アプリケーション開発中には、クエリプランを見て妥当なプランで動いているか、インデックスを使っているかなどを確認しチューニングを行なう。しかし、本番稼動後にいつも良いクエリプラ

ンで動いているとは限らない。運用が始まり、データ分布の変化やサンプリングで作成された統計情報から非効率なクエリプランが選択される場合や、開発時とは違うクエリプランで動いていることがあるため注意が必要である。

クエリプランの確認にはSETステートメントを使った方法とSQL Traceやプロファイラを使う方法があるが、ここではSETステートメントを利用した方法を紹介する。**表11**に実行プランに関わるSETステートメントの種類を示す。開発時にクエリプランを確認している場合は、そのプランを保存しておくことで本番稼動後にプランが変化したことを確認できる。

SQL Serverには、推定プランと実際のプランがある。クエリ実行時点で作成されるのが推定プランであり、クエリ実行後に生成されたクエリプランが実際のプランである。推定プランと実際のプランとは違う場合があるため、クエリが遅い場合には実際のプランを参照することをお勧めする。しかし、クエリが長時間終わらない場合の分析には推定プランからアプローチする必要がある。

XML形式でプランを取得した場合は、XMLデータに「.sqlplan」という拡張子を付けて保存し、管理ツールのSQL Server Management Studioから読み込むことでグラフィカルに表示できる。**LIST3**にSQL Serverのサンプルデータベースである AdventureWorksLTで実行したサンプルクエリを、**図9**にグラフィカル表示したサンプルクエリのプランを示す。

先述したように、クエリが非効率なプランで動作しているときはFULLSCANオプションを使った統計情報更新が有効である。しかし、統計情報を適切に更新してもクエリが遅い場合がある。そのときはクエリヒントを利用する。クエリヒントはクエリに結合方法やインデックスの使用、クエリ実行ごとのコンパイルなどを強制できる。**表12**にクエリヒントの一部を示す。クエリヒントはWITH句やOPTION句としてクエリに適用される。

SQL Serverは柔軟に最適なクエリプランを選択するため、クエリヒントの利用はどうしても効率的なプランで動作しないなど、やむを得ないケースでのみ利用することをお勧めする。LIST3のクエリにインデックスのシークを強制するヒントを付与した際のプランを**図10**に示す。

*　　*　　*

表11：SETステートメントでクエリプランを確認する方法

取得できるプラン	ステートメント	概要
推定プラン	SET SHOWPLAN_ALL ON	プランを含む各種実行情報をテキスト形式で出力する
	SET SHOWPLAN_TEXT ON	SHOWPLAN_ALLのサブセット
	SET SHOWPLAN_XML ON	実行情報をXML形式で表示する
実際のプラン	SET STATISTICS PROFILE ON	クエリを実行し、その実行情報をテキストで出力する
	SET STATISTICS XML ON	クエリを実行し、その実行情報をXML形式で出力する

表12：クエリヒントの種類と効果の一部。ほかにもたくさんのヒントがある。詳細はhttp://technet.microsoft.com/ja-jp/library/ms181714.aspxを参照

句	指定する引数	効果
OPTION	LOOP/MERGE/HASH JOIN	結合操作を指定する
	HASH/MERGET/CONCAT UNION	UNION操作を指定する
	MAXDOP	クエリ並行実行の最大数を指定する。 max degree of parallelismを上書きする
	OPTIMIZE FOR	クエリを特定の値に対して最適化する
	RECOMPILE	クエリプランの再利用をせず、実行ごとにコンパイルする
WITH	INDEX	特定のインデックスを利用させる
	FORCESEEK	テーブルやビューに対し、インデックスシークのみ利用させる
	NOLOCK	READUNCOMMITTED分離レベルと同じテーブルアクセスを行なう
	PAGLOCK	ロックの粒度をページ単位に行なう
	ROWLOCK	ロックの粒度を行単位に行なう
	SERIALIZABLE	SERIALIZABLE分離レベルと同じテーブルアクセスを行なう

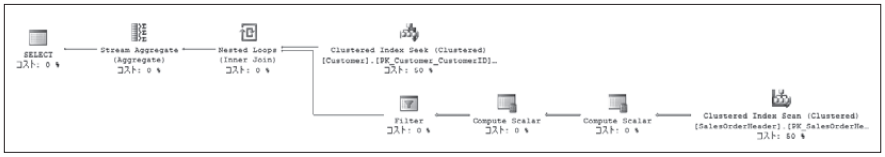


図9：XML形式で保存し、グラフィカル表示したプラン

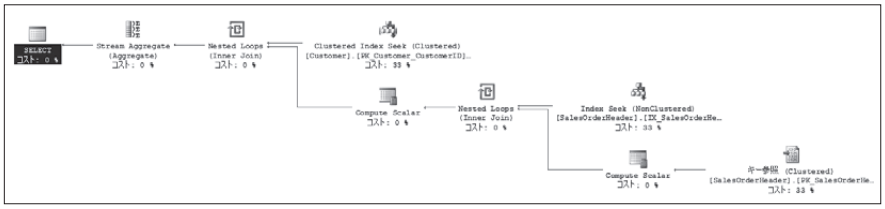


図10：クエリヒントでインデックスのシークを強制したプラン。元々Clustered Index Scanだったクエリプランが Index Seek(Non Clustered)とキー参照に変化していることが分かる

本特集では、SQL Serverが利用するWindowsの機能と、SQL Serverのテスト／設定／運用における注意点について紹介した。SQL Serverが利用するWindowsの機能には、今回は紹介できなかったカーネルモードでの動作やドライバの実装など、数多くある。これらの機能についてさらに深く知りたい方は、本特集を入り口に専門知識を学んでほしい。

また、SQL Serverが安定稼動を続けるためには、稼動前に問題を洗い出すこと、稼動後には発生した現象を解析できる資料を定常的に採取しておくことが重要となる。今回紹介している項目の中で皆さんが管理しているSQL Serverのデータベース上で確認したことがないものがあれば、ぜひ一度チェックして問題がないか検討してほしい。 **DBM**

内ヶ島暢之(うちがしまのぶゆき)
ユニアデックス株式会社 Windowsサポート部主任。入社以来リレーショナルデータベース担当。現在はミッションクリティカルサポートやSQL Serverの技術支援を行なっている。マイクロソフトと共同で仕事をすることも多く、SQL Server 2008 R2 早期実証プロジェクト(CQI)では自社のPMを担当した。