

IBM® Lotus® Notes®開発者のためのパフォーマンスの基本

[Andre Guirard](#)

Technical Solution Architect, Lotus Notes

IBM Software Group, WPLC

Minneapolis, MN

2008年5月

© Copyright International Business Machines Corporation 2008. All rights reserved.

このホワイト・ペーパーでは、IBM® Lotus® Notes®/Domino®アプリケーションのパフォーマンスに影響を与える最も重要で深刻な要因について説明します。本書は、Lotus Notes クライアント・アプリケーションの開発者を対象とし、問の領域を特定してソリューションを提供することにより、最大のパフォーマンスが得られるように支援します。

Lotus Notes 設計要素の作成方法、およびフォーム、フィールド、ビュー、列などのオプションの設定方法をすでに知っていることを前提とします。専門知識を持つあらゆるレベルの開発者に向けた内容が記載されています。

目次

1 はじめに	2
2 一般原則	3
3 データベース・レベルでのパフォーマンスの考慮事	5
4 式のパフォーマンス	7
4.1 @DbLookup と@DbColumn	8
5 フォームの設計	13
5.1 表示用の計算結果が機能するときは計算結果フィールドを使用しない	13
5.2 数百のフィールドを使用する	14
5.3 過度にグラフィックを使用する	15
5.4 フォームの格納	16
5.5 フィールドの自動更新.....	16
5.6 多すぎる共有設計要素	16
6 ビュー	17
6.1 ビューの選択式で@Now または@Today を使用する	17
6.2 不要なビュー	19

6.3 個人ビュー	21
6.4 不要な再ソート	21
6.5 不要な列	21
6.6 非常に複雑な式	21
6.7 複数のカテゴリ化の使い過ぎ	22
6.8 過度の索引作成	23
6.9 読者フィールド	24
6.10 最初は個人共有ビュー	26
7 コード	26
7.1 GetNthDocument	27
7.2 フォームまたはビューのアクション・コードが多すぎる	27
7.3 スクリプト・ライブラリーが多すぎる	27
7.4 ComputeWithForm	28
7.5 自動更新ビュー	28
7.6 効率のよいコレクション・ベースのメソッドの使用に失敗.....	28
7.7 高負荷のオペレーションの繰り返し.....	29
7.8 変更されていない文書の保存	29
7.9 文書の検索方法.....	30
7.10 使用されていない文書をキャッシュから削除する	31
7.11 より効率の高いループ、代入など.....	31
7.12 LC LSX の使用	32
8 テスト.....	32
9 プロフィール文書の使用	33
10 まとめ	33
11 リソース	33
12 著者について	34
13 謝辞.....	34

1 はじめに

Lotus Notes で簡単なアプリケーションを開発することは容易であり、ユーザーが少なく、文書もあまり多くない場合、パフォーマンスの問題が生じる可能性は低いです。しかし、アプリケーションが成功すると、ユーザーが多くなり大量のデータが蓄積していくでしょう。パフォーマンスを考慮せずにアプリケーションを設計していた場合、アプリケーションのレスポンスは大幅に低下します。

このホワイト・ペーパーでは、Lotus Notes/Domino®アプリケーションのパフォーマンスに影響を与える主な要因を取り上げ、パフォーマンスを最大限に高めるために開発者ができることについて説明します。本書はすべてを網羅したガイドではなく、むしろ最も一般的で最も深刻な設計の問題に焦点を当てます。

本書の目的は、主に Lotus Notes クライアント・アプリケーションについて、問題の領域を特定できるよう開発者を支援し、ソリューションの方向を示すことです。Web アプリケーションは同じような設計上の問題を抱えていますが、パフォーマンスについてさらに考慮が必要な事項や機会があります。これらについては、IBM Redbooks 資料「[Domino アプリケーションのパフォーマンスに関する考慮事項](#)」の付録 C 群または IBM Business Partner 文書「[Performance Engineering Notes/Domino Applications](#)」(US)を参照してください。

2 一般原則

一般に、以下に示す要因はアプリケーションのパフォーマンスに最も大きな影響を与えます。

- **ビューの数と複雑さ。** 使用されていないビューを削除するか、類似するビューをマージします。可能であれば、再ソート可能な列を使用して、同じ文書を持ちソート方法が異なる複数のビューを1つにまとめます。不要な列を削除し、選択式およびビューの列式を簡素化します。自分がアクセス権を持たない可能性があるサーバー上の個人ビューおよび他のビューをチェックします。
- **ビューの選択式または列式での@Today および@Now の使用。** 可能であれば使用を避けます。IBM Support Web サイトの Techdoc「[Time/Date views in Notes: What are the options?](#)」(US)、および本書の「ビュー」セクションを参照してください。
- **文書の数。** 文書数が多いほど、ビューを開く速度が遅くなります。古い文書をアーカイブするか、主要文書と返答文書を1つの文書にまとめることを考慮してください。たとえば、主要文書が「注文」の場合、各品目を別文書にするのは良くないアイデアです。Lotus Notes はリレーショナル・データベースではなく、文書指向のデータベースです。
- **文書に格納された要約フィールドの数。** リッチテキストではない各フィールドを「要約」フィールドと呼びます(かなり単純化した表現です)。文書内の要約フィールド数

が多くなるほど、ビューへの索引作成の時間がかかるようになります(数百のフィールドがある場合、最大約 30%まで増加します)。これは、ビュー内で要約フィールドが使用されていない場合にもあてはまります。文書数を減らそうとして、より多くのフィールドが必要になることがあります(この逆の場合もあります)。正しい選択を行い、最良のパフォーマンスを得るには、考慮が必要です。

- **フォームの複雑さ。** フォーム上で、フィールド数を実際に必要な数に制限することを試みます。長いフォームは、開いたり、更新および保存したりする際に、かなり時間がかかります(また、ビューのインデクサーが処理するフィールド数も増えることとなります)。
- **文書の変更。** 文書を不必要に変更すると、インデクサーの処理が増えてビューの索引作成が遅くなり、複製と全文索引の作成も遅くなります。
- **削除した文書の数。** 文書を削除すると、その跡に「削除スタブ」というマーカーが残されます。複製プログラムは、もう一方のレプリカから同じ文書を削除するか、失われた文書をこのレプリカにコピーするかを判断するために、このマーカーを必要とします。最終的に削除スタブは古くなり期限切れとなるため(デフォルトは 120 日)、通常のデータベースでは、問題になるほど削除スタブは蓄積しません。

しかし、文書数よりも何倍も多くの削除スタブがあるアプリケーションを見かけることがあります。通常、このようなことが発生するのは、すべての文書を削除する夜間実行のエージェントがあり、その後、外部のデータ・ソースからすべての新規文書が作成されるケースです。このようなことは行わないでください。ソース・データとの間で文書を比較し、どちらを更新または削除すべきかを判断するより高度なアルゴリズムを利用できます。詳細については、この [Lotus Sandbox のダウンロード\(US\)](#) を参照してください。

- **読者フィールド。** 読者フィールドは、必要がある場合に使用します。同等のセキュリティー・レベルを得られる別の方法はありません。しかし、多数の文書があり、そのほんの一部にしかユーザーがアクセスできない場合は、ビューでのパフォーマンスへの影響を考慮してください。影響を最小限にするヒントについては、本書の「ビュー」セクションおよび [developerWorks](#) の記事「[Lotus Notes/Domino 7 アプリケーションのパフォーマンス:第 2 部:データベースのビューの最適化](#)」を参照してください。
- **ユーザーの数。** 1 つのサーバー上に多数のユーザーがいると、アプリケーションおよ

びサーバーのパフォーマンスが低下します。また、アプリケーションのパフォーマンスがすでに限界に近い場合は、ユーザーを追加するとパフォーマンスはさらに悪化します。設計上の問題を修正することによって改善する可能性はありますが、特にクラスター・サーバーなど他のサーバー上にレプリカを作成するか、より高速なローカル・レプリカを作成するようユーザーに推奨するとよいでしょう。

3 データベース・レベルでのパフォーマンスの考慮事項

パフォーマンスの調整のために使用できるデータベース・オプションの一覧については、Lotus Domino Designer ヘルプの文書「データベースのパフォーマンスを向上するプロパティ」を参照してください。ほとんどのケースで、これらのオプションはパフォーマンスと機能の引き換えとなるので、特定のアプリケーションで機能が不要な場合は、その機能を無効にします。

最も顕著な影響を持つ可能性があるオプションは以下のとおりです。

- 「未読マークを使用しない」。
- 「LastAccessed プロパティを保持」。保持しない場合は、文書が最後に読み込まれた日時を知ることができません。この情報は、しばらく読まれていない文書をアーカイブするとき便利です。
- 「文書の階層情報を使用しない」。階層情報を無効にすると、@AllDescendants または@AllResponses を使用できません。これらの@関数は、ビューの選択式および複製式で役に立つ場合があります。
- 「トランザクション・ログを無効」。このオプションがパフォーマンスに与える効果は、システム管理者がサーバー上で設定した内容やユーザー数に応じて異なります。ユーザー数が多い場合、トランザクション・ログを使用すると、使用しない場合よりも速くなるでしょう。2つの方法を試し、測定を試みてください。トランザクション・ログは、復旧に使用されます。
- 「文書テーブルマップの最適化」。このオプションは、アプリケーションにほぼ同数の異なるタイプの文書があり、ほとんどのビューで 1 つの文書タイプ(たとえば、SELECT Form = “xyz” & ... など)だけが表示される状況で最も役に立ちます。このように、最初にフォームの確認をするようビューの選択式を記述すると、そのフォームを使用していない各文書をすぐに対象外にできるため、ビューの索引作成が速くなります。

NSFDB2 (Lotus Domino データを DB2®データベースに格納する機能)を使用してもパフ

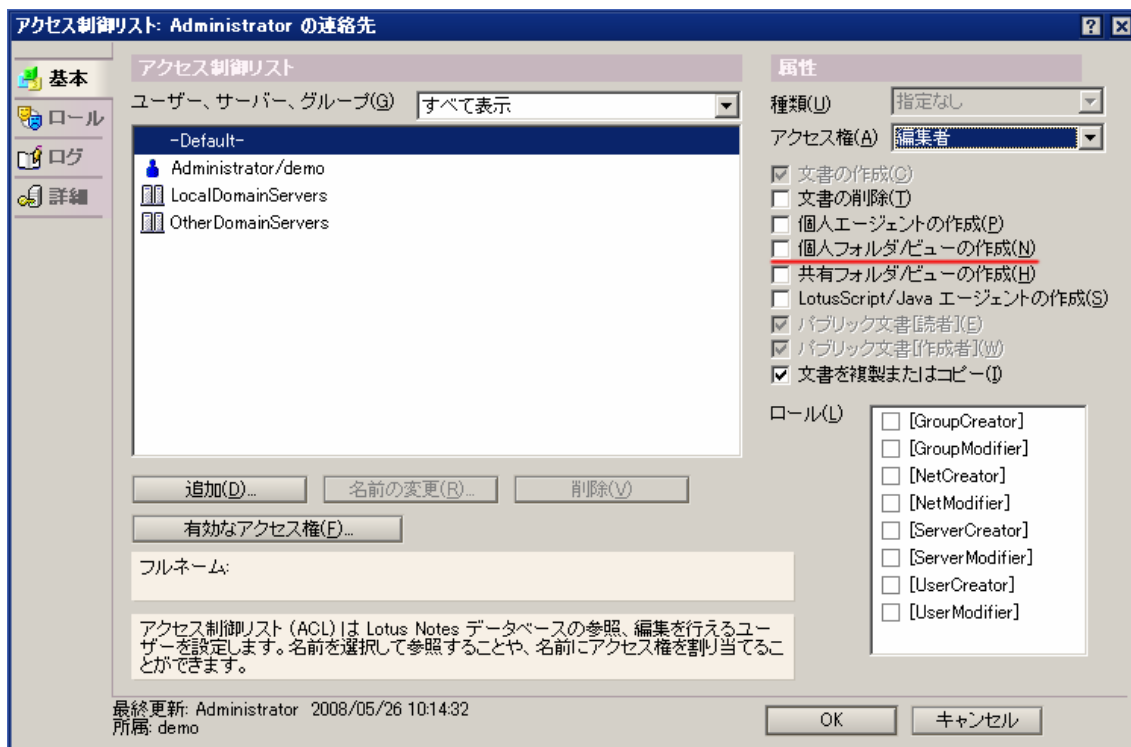
パフォーマンスには貢献せず、実際には、従来の NSF ファイルを使用するよりも通常はやや遅くなります。NSFDB2 の目的は、パフォーマンスの向上ではなく、機能性を追加することです。

全文索引は多くのディスクスペースを占める可能性があります。通常はその価値があります。この機能を活用するとエージェントでより速い検索を実行でき、索引がない場合、ユーザーはサーバーに負荷をかける遅い検索方法の使用を強いられ、結果を得るまでに長い時間がかかります。

メモ: Lotus Notes 8.0 の新しいデータベース・プロパティを使用すると、データベースに全文索引が作成されていないときに、全文検索を無効にできます。一般に、全文索引がある場合でも、誤って索引が削除されたときに、説明なしに急激にパフォーマンスが低下するよりも、ユーザーにメッセージを確実に表示する方がよいでしょう。

プロパティ・ボックスの他に、データベース・レベルで実行できるもう 1 つのことは、ACL ダイアログ・ボックスにあります。個人ビューおよび個人フォルダを作成するユーザーのアクセス権を制限することにより、サーバーの負荷を削減できます(図 1 参照)。

図 1. ACL ダイアログ・ボックス



「個人フォルダ/ビューの作成」ボックスのチェック・マークを外すと、ユーザーは個人ビューを作成できることには変わりありませんが、ビューはサーバーではなく、ローカルのデスクトップ・ファイルに保存されます。このため、アプリケーションのパフォーマンスにはそれほど影響を与えません。

デスクトップの個人ビューは、索引を作成するために、サーバーからリアルタイムにデータを取得しなければならないので、パフォーマンスに影響する可能性があります。このため、デスクトップの個人ビューを多用することも、サーバーのダウンにつながる可能性があります。したがって、ビューのオプション「最初は個人」を使用してユーザー用に個人ビューを自動的に作成することは避けてください(この内容については、後で詳しく説明します)。

4 式のパフォーマンス

ほとんどの@関数はかなり高速に実行されますが、評価に時間がかかるものもあります。以下の@関数に注意し、効率よく使用してください。

- **@Contains** はあまり負荷の高い関数ではありませんが、リストに一致する値が含まれるかどうか確認するために使用されることがあります。この使い方は非効率的であり、

誤っています。たとえば、`@Contains(Cities; "Lansing")`という式は、`Cities`に”East Lansing”という値が含まれていると、`True` を返します。これが目的であれば問題ありませんが、“Lansing”という値に正確に一致するエントリーを検索する場合は、代わりに「=」、「*=」、または「@IsMember」を使用します。これらは、最初の文字が一致しない場合に文字列全体をスキャンする必要がないため、速くなります。

- **@For** および **@While** は、より効率の高い **@Transform**、またはリスト全体に一度に作用する他の関数で置き換えることができます。
- **@Unique**。この関数はリスト内の各値をそれ以外の値と比較するため、実行時間はリスト内のアイテム数の二乗に比例します。すでに値の重複がない状態となっているリストを取得した方がよいでしょう。これについては後述します。
- **@NameLookup** は **@DbLookup** に似ていますが、ディレクトリー情報にのみ使用されます。
- **@DbLookup**、**@DbColumn**。これらの関数を過度に使用または誤用すると、フォームを遅延させる大半の原因となるため、別のセクションをもうけて説明します(以下参照)。

マクロ言語のループ関数が不必要に使用されることがしばしばあります。Lotus Domino Designer ヘルプの文書では触れられていませんが、文字列引数を受け取るほとんどすべてのマクロ関数は、リストに対して実行することができます。たとえば、`x` がリストのとき、`@Left(x; “, ”)`は各要素に“Left”を実行したリストを返します。

メモ:以前は、`@UserRoles` および `@UserNamesList` はパフォーマンスを大きく低下させる原因となっていました。しかし、Lotus Notes 6.0 からこれらの関数の結果がキャッシュされるようになりました。

4.1 @DbLookup と@DbColumn

`@DbLookup` と `@DbColumn` のパフォーマンスに影響する 3つの主要因は以下のとおりです。

- キャッシュを使用する
- 検索するビューが効率的である

- これらの関数を不必要に使用する

4.1.1 キャッシュの使用

多くの開発者は、特にキーワード式で“NoCache”オプションを過度に使用しています。なぜ、このようなことが起きるのかを理解するのは簡単です。開発および初期テスト時は、キーワードが頻繁に編集される傾向があり、NoCache にしておくのと正しく動作するからです。

しかし、運用に入ると、アプリケーションのキーワードは毎日編集されないのが普通です。新しい値がユーザーに利用可能になるまでに若干の遅れが生じても、より良いパフォーマンスとの引き換えとして受け入れられます。NoCache は、本当に必要なときのみ使用してください。

キャッシュのオプションには、次の 3 つがあります。

- “Cache” (デフォルト)は、アプリケーション・セッションでユーザーが最初に特定の検索を行ったビューだけを参照し、アプリケーションが終了されるまで、その検索結果を記憶していて以降の検索に使用します。
- “NoCache”はキャッシュを回避し、常にビューを検索します。同じ検索にキャッシュされた値が存在する場合でも、その値は更新されません。
- “ReCache”は忘れられたオプションですが、常にビューを検索します。ただし、検索値でキャッシュを更新します。ReCache を使用すると、意図的にキャッシュを特定の時間、たとえば検索が参照する文書が保存されたときに更新できます。それ以外のときは、キャッシュされた値を使用できます。このキャッシュ値は、ユーザーが入力した情報により、少なくとも最新の状態であることがわかっているためです。

4.1.2 検索用の正しいビューの選択

効率の良いビューであっても、@Db 関数用に最適の選択ではない場合もあります。たとえば、@Unique(@DbColumn(“”:“NoCache”; “”:“”; “InvoicesByCompany”; 1))には複数の問題があります。

- 不要であるかもしれない NoCache を使用しています。毎日会社名を追加するわけではありません。追加するときは、Invoice フォームの Postsave で“ReCache”オプションを使用して、新しい会社名をすぐに利用可能にできます。
- 現在のデータベースが“”:“”で指定されています。“”:“”は句読点と紛らわしいだけでなく、評価も少し遅いため、代わりに“”を使用してください。

- 重複する値を含むリストを検索せず、@Unique を使用して重複を解消します。あるいは、カテゴリー化された列により、値の重複がないビュー列を検索します。

この最後のポイントが特に重要です。テスト・データの 100 文書を使用して列の検索が正しく機能しても、アプリケーションの運用を開始すると文書数が数千にも増え、すぐに検索速度が低下してしまいます。特にアプリケーションがサーバー上で使用されている場合は、ビュー列のすべての内容をネットワーク経由でユーザーのクライアントに送信するまでに、かなりの時間がかかります。すでに固有の値をビューから読み込む方が、大幅に速くなります。

メモ:「索引にユニークなキーを作成する」オプションは、固有の値を得るためにカテゴリー化された列の代わりに使用できそうですが、このオプションには欠点があり、そのような使い方には適していません。

また、索引作成に時間がかかるビューを検索することも良いアイデアではありません。特に、選択式や列式で@Today または@Now が使用されているビューの検索は良くありません。特定の日付を持つ文書の検索だけが必要な場合は、@DbColumn を使用してこのような文書のみ含まれるビューを検索するのではなく、@DbLookup を使用して、日付でソートされたすべての文書が含まれるビューを検索します。このとき、日付を検索キーとして指定します。

4.1.3 繰り返し検索の防止

不必要に@Db 関数を使用するケースは、さまざまな状況で発生します。共通するいくつかの例を紹介しましょう。

式内での繰り返し

```
@If(@IsError(@DbLookup(“”: “NoCache”; “”; “SomeView”; CustID; 3);
“”);
@DbLookup(“”: “NoCache”; “”; “SomeView”; CustID; 3))
```

この式は、必要ないかもしれない NoCache を使用しているだけでなく、1 回の検索でよいところを 2 回検索しています。代替の式を 2 つ示します。

```
_tmp := @DbLookup(“”; “”; “SomeView”; CustID; 3);
@If(@IsError(_tmp); “”; _tmp)
```

または

```
@DbLookup(“”; “”; “SomeView”; CustID; 3; [FailSilent])
```

読み込みモードでの不要なキーワード検索

文書が表示用に開かれているとき、特定のタイプのキーワード・フィールドに対し、Lotus Notes は選択肢のリストを認識する必要がありません。明らかな例外は、チェック・ボックスおよびラジオ・ボタンのフィールドと(これらのフィールドには、読み込みモードでもすべての選択肢が表示されます)、キーワード・シノニム(“表示テキスト|値”)を使用するすべての要素です。これは、文書は“値”のみ格納するのに対し、フォームは“表示テキスト”を認識している必要があるからです。

別のケースとしては、実際に選択肢のリストが必要になるまで、検索を遅らせるキーワード式を記述します。

```
_t := @If(@IsDocBeingEdited; @DbColumn(“”; “”; "Customers"; 1);  
@Return(@Unavailable));
```

```
@If(@IsError(_t); “”; _t)
```

この式は、文書が読み込みモードのときに@Unavailable を返すことにより、後で選択肢のリストが必要になったときにもう一度要求するようフォームに指示しています。これは、ユーザーが編集モードに切り替えたときに発生し、カーソルが選択フィールド内に置かれます。

つまり、ユーザーが単に文書を見ているときは検索を避けるだけでなく、文書の編集時に遅延を分散させます。0.5 秒の遅れ 8 回の方が、4 秒の遅れ 1 回よりも、かなり煩わしさが少なくなります。また、ユーザーがそのフィールドにカーソルを移動させない場合、検索を待つ必要はまったくありません。

1 回の検索でよいときに複数回の検索を行う

顧客の ID が「請求書」文書に格納されていて、この ID を使用して顧客の名前、住所、および購買担当者名を検索して表示するケースを考えましょう。フォーム上にいくつかの表示用の計算結果フィールドがあり、各フィールドには@DbLookup(“”; “”; “CompanyByID”; CustID; x)を使用した式が設定されています。x は列番号またはフィールド名です。

必要なすべての値を含む 1 つの列を作成し、それを分離して個々のフィールド値を得る方が効率的です。この列式は次のようになります。

`CustName : StreetAddress : (City + " " + State + " " + Zip) : PurchasingContact`

フォーム上で、`CustDetails` という名前の表示用の計算結果フィールドを 1 つ作成します。このフィールドは非表示にします。

`@DbLookup(""; ""; "CompanyByID"; CustID; 4)`

この場合、結合した列の列番号を 4 としています。そして、名前を表示したいところで次の式を使用します。

`CustDetails[1]`

以降、同様にします。

更新時に検索を繰り返す

フォームの作成時に、ユーザーのマネージャー名を計算結果フィールドで検索するケースを考えます。次の式を使用します。

`@DbLookup(""; "VOLE1"; "EmpData.nsf"; "EmpByName"; @Name([CN]; @Username); "Manager")`

計算結果フィールドは、文書が更新されるたびに再計算されます。「キーワードの変更時にフィールドを更新」オプションを有効にすると多くの文書が頻繁に更新されるため、大きな速度低下につながります。フィールドを「作成時の計算結果」にするとよいでしょう。

フィールドを文書に格納する必要がない場合は(格納する必要がないフィールドは、格納しないようにしてください)、「表示用の計算結果」にすることもできます。しかし、この場合は、次の方法で更新時の検索の繰り返しを防止してください。

`@If(@IsDocBeingLoaded;`

`@DbLookup(""; "VOLE1"; "EmpData.nsf"; "EmpByName"; @Name([CN];`

`@Username); "Manager");`

@ThisValue)

@DbColumn を使用して連続番号を割り当てる

これは頻繁に行われる誤りです。設計者が各文書に固有の ID を作成しなければならないとき、既存文書の最後に“1”の番号を追加することがよくあります。次のような式を記述します。

```
tmp := @DbColumn(“”:“NoCache”; “”:“RequestsByNumber”; 1);
nextNumber := @If(tmp = “”: 1; @ToNumber(@Subset(tmp; -1)) + 1);
@Right(“000000” + @Text(nextNumber); 7)
```

しかし、これは実際には良くないアイデアです。文書の数が増えるほど、@DbColumn は実行に時間がかかるようになります。さらに、アプリケーションに複数のユーザーがいるとき、特に複数のレプリカがある場合、実際にこの方法では固有の ID が保証されません。

文書の保存時に番号が割り当てられるのでは、その時点まで番号が利用できず不便です。一方、文書の作成時に番号を割り当てると、他のユーザーが同じ番号で文書を作成して保存するのに十分な時間が生じてしまいます。

要件をもう一度見直してみましょう。人々が連続した番号付けを求めている場合でも、実際に、アプリケーションに必要なのは固有の識別子であって、数値である必要はない場合があります。@Unique 関数を調べると、この関数はほとんど固有とみなせる適度に短い値を生成します(固有であることは、少し追加作業をするだけで保証できます。たとえば、各ユーザーにイニシャルなどの固有の接尾辞を割り当てます)。

本当に連続番号が必要な場合は、[developerWorks](#) の記事「[複製アプリケーションで順序番号を生成する](#)」に、かなり効率のよい1つの方法が記載されているので参照してください。この件については、今後の [developerWorks](#) の記事も参照してください。

5 フォームの設計

このセクションでは、注意する必要があるいくつかの一般的な問題について説明します。

5.1 表示用の計算結果が機能するときは計算結果フィールドを使用しない

一般に、フィールドを格納するとアプリケーションの速度が低下するので、必要なときに容易に計算できる値は格納しない方が効果的です。次のような代替手法があります。計算結果フィールドは、文書を読み込みモードで開くときには計算されません。このため、低

速の式を格納することにより、読み込みモードのパフォーマンスを改善できます(一方、情報が古くなる可能性があります)。

しかし、他のフィールドの値を単に再表示するだけのために計算結果フィールドを使用しないでください。使用すると、同じ情報の2つのコピーを保存することになります。

5.2 数百のフィールドを使用する

1つのフォームに大量のフィールドが配置される最も一般的な理由として、複数の行と列で情報を示す表があり、各セルに1つのフィールドが含まれ、サポートできる最大の行数に達しているケースが挙げられます。表を使用すると、このような機能をとっても簡単に提供できるため、これは非常に厄介な状況です。

一方で、値の表を管理する別の方法もあります。最も明快な方法としては、実際に表をリッチテキスト・フィールドに配置し、ユーザーに自由に入力してもらいます(リッチテキスト・フィールドのデフォルト式で@GetProfileFieldを使用し、オープン時に表をプロフィール文書から読み込みます)。この方法の欠点は、個々のフィールドがあれば提供できるキーワード・リスト、入力変換、入力確認などの入力補助を、ユーザーがセル内でまったく利用できないことです。しかし、代替手法としてこれを許容できる場合もあります。

また、ダイアログ・ボックス内の表で一度に1つの行を編集し、結果を表内に表示する手法やツールもいくつか公開されています。たとえば、Lotus Sandboxの[Domino Design Library Examples\(US\)](#)に、セルごとにフィールドを持たせずに表内のデータを編集や表示することができる設計要素のセットが含まれています。このシステムの詳細はドキュメント・データベースの「Table Editor」という文書(US)に記載されています。インプリメントするために多少の作業が必要ですが、パフォーマンスには非常に効果があります。

ほとんどの文書で、多くのフィールドがブランクのまま残されているフォームを見ることがあります。たとえば、50フィールドを持つ「規制認可」セクションが必要となる文書は5%ほどでしょう。残りの95%では、ブランクのフィールドがすべて格納され、スペースの無駄が生じるとともに、パフォーマンスが低下します。

このようなケースでは、2つのフォームを作成するとよいでしょう。1つは、常に必要なフィールドを含むメイン・フォームです。もう1つは「規制認可」フォームで、元の文書への返答として必要な場合にのみ作成します。これは、余分な多数のフィールドを持つことを避けるために、追加文書を用意する方がよいケースです。

複数値フィールドも忘れてはなりません。5つのフィールドを配置して最大5つの値までユーザーに入力してもらうよりも、複数値を入力できる1つのフィールドを使用します。このようにすると、入力値の数に制限はなく(1つに限定しない限り)、結果として得られる値は、ビューおよび式ではるかに使いやすくなります。

メモ:多数のフィールドにより既にアプリケーションが遅い場合、設計要素を変更するだけでは速度の上昇にほとんど寄与しません。既存のすべての文書に対して実行するエージェントを作成し、内部の余計なアイテムを削除する必要があります。これを簡単に行うことができるビジネス・パートナー製品があります。しかし、いくつかのフィールドを専用の返答文書に移すなど、変更に必要な再編成がともなう場合、これらのエージェントは非常に複雑になります。先のことを見通して、初めからきちんと対処することにより時間を節約できます。

5.3 過度にグラフィックを使用する

背景用の大きいビットマップや多数の装飾用の仕掛けなど、極度にグラフィックを使用しているフォームがあります。大きいイメージは、ロードに時間がかかり、設計要素のキャッシュ内でメモリーを占有し、フォームを表示するときに描画時間がかかります。フォームの作成時に少し気を配るだけで、パフォーマンスに大きな影響を与えずに、プロフェッショナルな外観が得られます。いくつかのヒントを紹介します。

- フォーム上にイメージを貼り付けないでください。代わりに、イメージ・リソース設計要素を使用するか、イメージをインポートします。複数のフォームで同じイメージを使用する予定のときは、イメージ・リソースを使用してください。イメージ・リソースにより、クライアントはフォームの設計とは別にイメージをキャッシュするようになります。複数のフォームで同じイメージを使用する予定がない場合でも、後で他の開発者が同じイメージを使用して別のフォームを作成する可能性もあるので、イメージはイメージ・リソースにしておくといでしょう。
- イメージをフォーム上に配置した後で、使用するサイズに縮小しないでください。グラフィック・エディター(たとえば、GIMP)を使用して、オリジナルの大きいイメージを必要なサイズに縮小してください。たとえ、同じイメージについて異なるサイズのイメージ・リソースが複数必要となった場合も、このようにしてください。

イメージがJPEGの場合は、縮小時に異なる圧縮設定によってファイルのサイズを削減できないか試してください。JPEG圧縮は劣化をともなうので、圧縮するとオリジナルのイメージの品質は保てません。しかし、外見的に劣化していないように見える

範囲で圧縮すると、フォームのロードが速くなります。このバランス・ポイントの検出を支援する市販のツールがあります。

- イメージに適した正しいファイル形式を使用してください。イメージに制限されたパレットを使用する場合(大抵のロゴがそうであるように)、通常は GIF 形式にすると、最も小さいファイルが生成されます。フルカラーの写真または絵の場合は、通常 JPEG がベストです。BMP ファイルは、通常はまったく圧縮されていないので絶対に使用しないでください。
- 表のセルやそのセルの背景のグラフィックは、描画に時間がかかります。特に 3D 効果を持つ境界線など、セルの境界線を非表示にすると、表示するときよりもレンダリングが速くなります。マージされたセルを持つ表は、他の表内にネストされた表よりも速くレンダリングされます。

5.4 フォームの格納

文書へのフォームの格納は使用しないでください。

5.5 フィールドの自動更新

フォームのオプション「フィールドを自動更新」は、できるだけ使用しないでください。これを使用すると編集時にフォームが頻繁に更新され、計算結果フィールドおよび入力変換式の再計算時に遅延が生じます。通常は、フィールド・レベルのオプション「キーワードの変更時にフィールドを更新」またはフィールドの **Onchange** イベントや **Onblur** イベントを使用して、必要なときだけ更新します。

5.6 多すぎる共有設計要素

フォームは、イメージ・リソース、共有フィールド、共有アクション、サブフォーム、アウトライン、スタイルシート、スクリプト・ライブラリーなど、他の設計要素から情報を取得できます。1つの文書を開くときに、フォーム以外の多数の設計要素から情報を読み込むことは可能ですが、これには時間がかかります。共有設計要素の利点は、アプリケーションのメンテナンスを容易にすることです。欠点は、ロード時に複数のノートにアクセスし、時間がかかることです。

Lotus Notes は設計情報のキャッシュを維持しているので、毎回オリジナルの設計要素から設計情報を読み込む必要はありません。しかし、初期ロードの時間が問題になることがあります。また、同じサブフォームやイメージが多数のフォームで使用されている場合は、キャッシュにより、共有設計要素を使用することがパフォーマンスに寄与する可能性があります。

ります。

共有アクションが含まれる設計ノートは 1 つだけなので、共有アクションはパフォーマンスを損ねません。1 つ分の負荷で複数のアクションを使用できます。ビューの共有列もパフォーマンスに影響しません。

保守容易という利点があるので、他の方法を試してもパフォーマンスが依然として好ましくない場合のみ、設計要素の共有を解除することを推奨します

6 ビュー

非効率的で不要なビューは、以下の原因により遅延を生じさせます。

- ビューを開く際の、索引更新時間。
- @Db 関数でビュー使用時の、情報取得に関する時間。
- サーバー上の Update タスクは定期的に各ビューで最近変更された文書を確認して、ビューを更新する必要はないかを確認しています。このため、より多くのビュー(またはより複雑なビュー)はサーバーの負荷となり、すべてのアプリケーションの速度が低下します。

ビューを開くのが遅くなる他の一般的な原因として、データベース内の大量の文書が挙げられます。ユーザーがビューを開くとき、ビュー索引の最終更新以降に新たに変更された文書がないか Lotus Notes はチェックします。文書の数が多くなるほど、たとえ変更された文書がなくても、このチェック自体に時間がかかります。

6.1 ビューの選択式で@Now または@Today を使用する

@Today または@Now を使用せずに日付/時刻に基づくビューを作成する方法は、数多く書かれています。1 つの例として、IBM Support Web サイトの Techdoc「[Time/Date views in Notes: What are the options?](#)」(US)があります。これには、日付/時刻に基づくビューを作成する 1 つの方法が示されています。

より詳細な点についていくつか考えてみましょう。まず、よく繰り返されるアドバイスである@TextToTime(“Today”)の使用しても、完全に置き換えられるものではありません。単独では、この式は最初の日しか機能しません。これを正しく機能させるには、追加作業が必要です。

なぜでしょうか。通常、ビューを開くとき、Lotus Notes はビューの索引(ビュー内の文書および行の値を保存したリスト)を見て、最終更新以降に作成または変更された文書だけを調べ、ビューに追加するか、ビューから削除するか、または列の値を再計算するのかを判断します。ビューの最終更新以降に変更された文書がない場合、このプロセスは非常に高速です。

しかし、@Today を使用すると、古いビュー索引が役に立たなくなります。たとえば、次の選択式について考えましょう。

```
SELECT Status = "Processing" & DueDate <= @Today
```

@Today の値はビューが最後に使用された後に変更されたため、文書の内容が変わらないにもかかわらず、文書がこのビューに追加される可能性があります。このため、ユーザーがこのビューを使用するたびに Lotus Notes は古いビュー索引を破棄し、データベースの各文書をチェックして、それがこのビューに属しているかどうかを決定し、列の値を変更します。

@Today の代わりに@TextToTime("Today")を使用して、ビューのインデクサーの「裏をかく」ことができます。都合のよいことに、Lotus Notes は古いビュー索引を再使用して、変更された文書だけを調べます。これはより速く機能しますが、残念ながら正しい結果を得られません。@Today が変わると、すべての文書を再びチェックしなければならないからです。

たとえば、@Now と比較して 3 時間経過しても「要求」文書がまだ Open(未処理)のときに、赤色の感嘆符を表示する列があるものとします。この状況は、ビューが最後に使用されたのが 5 秒前でも、変わる可能性があります。@Today の使用は、ビュー索引をより長い間隔で更新するためにのみ適しています。

これを実際に行うには、ビューのプロパティ・ボックスのビュー索引のオプションを使用します。「詳細」タブで「自動(周期)」を選択し、「時間毎」に時間を指定することで、指定した周期でビューを更新できます。この利点は、ビューが非常に素早く開くことです。欠点は、文書に加えられた変更内容がすぐにはビューに表示されないことです。ユーザーは、最新のデータを見るために、手動でビューを更新する必要があります。

一般的なもう 1 つの代替手法として、夜間に実行する定期実行エージェントを作成し、ビューの選択式がその日の選択式を含むように更新する方法があります

(NotesView.SelectionFormula メソッドを使用します)。たとえば、エージェントは次のステートメントで実行できます。

```
view.SelectionFormula = {SELECT Status="Processing" & DueDate={} & Today & {}}
```

しかし、この方法にもいくつかの欠点があります。

- すべてのレプリカが正しい文書を表示する前に、ビューの設計変更をすべてに複製する必要があります。
- サーバー管理者が、アプリケーションの設計を変更するエージェントに疑いを持つ可能性があります。
- 翌朝、ビューを最初に開くユーザーは、ビューの索引が作成されるまで待たなければなりません。ビューの索引オプションを「自動」に設定するか、エージェントにビューを更新させることにより、この問題を回避できます。
- データベースが自身の設計をテンプレートから取得している場合は、ビュー設計がテンプレートによって上書きされます。これを回避するには、夜間の設計更新後にエージェントを実行するようスケジュールを変更するか、テンプレート自体に変更を加えます。

もう 1 つの解決策として、ユーザー・インターフェースで妥協する方法もあります。たとえば、「期限切れの未処理の要求」ビューの代わりに、「期限別の未処理の要求」ビューを作成し、ビューの一番上で期限切れ要求をソートできるようにします。このようにすると、期限切れの要求を簡単に発見でき、ビューもかなり速く開きます。

場合によっては、フォルダを使用し、日付条件に基づいて文書のグループを表示することが適していることもあります。夜間に実行するエージェントで、その日付に応じた文書を持つフォルダを作成し、フォルダのアクセス設定により、ユーザーがその内容を手動で変更できないようにすることが可能です。就業時に文書を編集することでフォルダの内容が変更される場合、この方法は適していません(カスタム・コーディングによってもこの方法を管理できますが、面倒です)。

6.2 不要なビュー

多くのアプリケーションは多数のビューを持っているために遅く、ビューを少しでも削除することが速度の向上に貢献します。一般に、これは特定のアプリケーションというより、サーバーのパフォーマンスに効果があります。

メモ:データベースの設計者は、必ずしも各ビューを見るためのアクセス権を持っているとは限りません。ユーザーが作成したサーバー上の個人ビュー、および開発者を含まない読者リストを持つ他のビューは表示されませんが、パフォーマンスに影響します。サーバー管理者は、「フルアクセス・アドミニストレーター」モードを使用してこれらのビューを見ることができます。

ビューのデフォルトの更新設定(「自動(最初の使用后)」、「削除(45 日間使用されないとき)」)は、45 日間使用されないビューの索引は廃棄され、サーバーによって自動的に更新されなくなることを意味します。この時点で、パフォーマンスへの影響は最小になります。しかし、アウトラインにビューを配置すると、誰かが正しいビューを探しているときに、偶然そのビューを使用してしまう可能性があります。

したがって、ユーザーが必要としているビュー、ユーザーの特定のタスク用に設計されたビュー、およびユーザーが迷わずに選択できるような名前を持つビューだけに絞り込むことにより、パフォーマンスを向上させるだけでなく、操作時のユーザーの負担を軽減することができます。

ビューが 1 回限りの特殊な使用のために作成され、そのビューの要求ユーザー、使用ユーザー、安全に削除できる時期が記録されていないことがよくあります。これらは、作成したユーザーだけに表示されるサーバー上の個人ビューである場合もありますが、パフォーマンスには影響します。このようなビューの作成を制限することで、パフォーマンスの維持に役立ちます(どのような個人ビューがあるか知りたい場合、サーバー管理者は「フルアクセス・アドミニストレーター」モードを使用してビューをリストアップできます)。

ビューの「コメント」フィールドを使用して、作成目的、使用者、および削除可能になる終了日を記述することを推奨します(これらの情報が既知の場合)。このようにすると、ビューが必要かどうか疑問であるときに、少なくともそれを誰に問えばよいのかを知ることができます。ビューを削除して誰かが抗議するかどうか確認したいときは、後で復元できるように、文書を持たない他のデータベースにそのビューをカット&ペーストします。

ソート方法だけが異なるビューがアプリケーションに含まれることが時々あります。一般に、このようなビューは、再ソート可能な列を持つ 1 つのビューにまとめる必要があります。再ソート可能な列を追加することのコストは大きいですが、それでも別の異なるビューを持つよりは小さくなります。

これは、Lotus Notes 8.0 の新しい列のオプション「最初の使用まで索引作成を遅延する」

を使用すると特に顕著です。このオプションは、ユーザーが要求するまで再ソートの索引作成を遅延します。このオプションにより、最初のユーザーには大きな遅れが生じますが、再ソートを誰も要求しない場合は、どのユーザーも良好なパフォーマンスが得られます。

6.3 個人ビュー

不要なビューを検索するとき、開発者はアプリケーションのすべてのビューを見られるわけではないことに注意が必要です。ユーザーがサーバー上に作成した個人ビュー、またはアクセス・リストに開発者が含まれていない共有ビューは、**Lotus Domino Designer** で見ることができません。しかし、これらのビューはパフォーマンスに影響を与えています。サーバー管理者は、「フルアクセス・アドミニストレーター」モードを使用してアクセス制御を回避し、開発者のためにすべてのビューのリストを取得できます(また、削除したいすべてのビューを削除できます)。

6.4 不要な再ソート

ユーザーからの要求があり次第、サーバーはビューの交互のソートを利用可能にするために余分な作業が必要となるので、再ソートは本当に役に立つ場合にのみ有効にしてください。昇順と降順は異なる 2 つの再ソートとしてカウントされるので、両方が真に必要な場合以外は、両方を同時に有効にしないでください。**Lotus Notes 8.0** では、再ソートが使用されるかどうかわからない場合、その列に「最初の使用まで索引作成を遅延する」オプションを使用します。

また、列ヘッダーをクリックして、既にソート済みの別のビューに移る方法もあり、余分なコストをかけずに再ソートの利便性を提供できます(もう一方のビューがすでに存在する場合)。

6.5 不要な列

フィールドごとに列を作成したくなりますが、これは行わないでください。ビューの情報は、ユーザーが実際にそのビューで見る必要があるものだけに制限します。このようにすると、画面の乱雑さが減り、アプリケーション全体が速くなるとともにストレージの使用量も削減されます。

6.6 非常に複雑な式

ビューの列式または選択式でループ関数(@For、@While、@Transform)が使用されている場合、またはコメントを除く式の長さが、たとえば 200 文字以上の場合は、式の簡素化を試みます。簡素化できないときは、ビューがフィールド名だけを参照できるように、式を

フォーム上の計算結果フィールドへ移すことを考慮します。この方法は、特に複数のビューで使用されている式で役立ちます。

計算結果フィールドによる方法を用いない場合でも、長い式の多くは、少し工夫するだけで簡素化できます。長い@If ステートメントの代わりに@Select または@Replace を使用することや、ロジックを見直して、順序を変更することでテストを簡素化できないか確認します。

リストのすべてのメンバーに適用される演算子および@関数には注意してください。文字列リストに多数の単純な操作を行うためにループを記述する必要はありません。たとえば、各要素の最初の 3 文字を得るには@Left(listfieldname; 3)を使用します。

また、*=のような「組み合わせ的な」演算子もあり、2つのリスト各要素の組み合わせを比較したり、より簡潔な式を記述したりするときに役立ちます。

他のプログラム言語でのプログラミング経験がある場合は、結合の値を決定するために必要な式だけを評価する論理演算子に慣れていることでしょう。たとえば、次のような式の場合です。

Form = "Report" & (Sections = "Financials" | Total > 10000)

この式では、まず **Form** が **Report** であるかどうかチェックし、これが真の場合にのみ、残りの式をテストする、という処理が期待されます。しかし、マクロ言語(および LotusScript)では、論理演算子はそのように機能しません。式のどちらの部分も常に評価されます。このため、2番目の部分の評価負荷が高い場合は、次のような「横着な」ロジックの式を選択することもできます。

@If(Form = "Report"; Sections = "Financials" | Total > 10000; @False)

@If 関数は&演算子よりも実行に時間がかかりますが、@If 関数を使用して負荷の高い関数が不必要に実行されるのを防ぐことは利点となります。

6.7 複数のカテゴリー化の使い過ぎ

カテゴリーは優れた機能です。カテゴリーを使用すると、同じビュー内で複数の見出しのもとに文書をリスト表示でき、たいへん有用です。しかし、ビュー内で文書のリストを 2 回表示することは、1回の表示のほぼ 2 倍の時間がかかるので、使い過ぎはよくありません。

もし、各文書が 50 のカテゴリに属していると、それに文書数を掛けることになり、サーバーが計算および格納しなければならない行数が膨大な数になります。

複数のカテゴリを使用しない場合でも、カテゴリ化されたビューは、単純なソートを設定した同じビューよりも速度が遅くなります。時間は文書数ではなく、行数に基づいており、各文書と各カテゴリの見出しは行とみなされます。

6.8 過度の索引作成

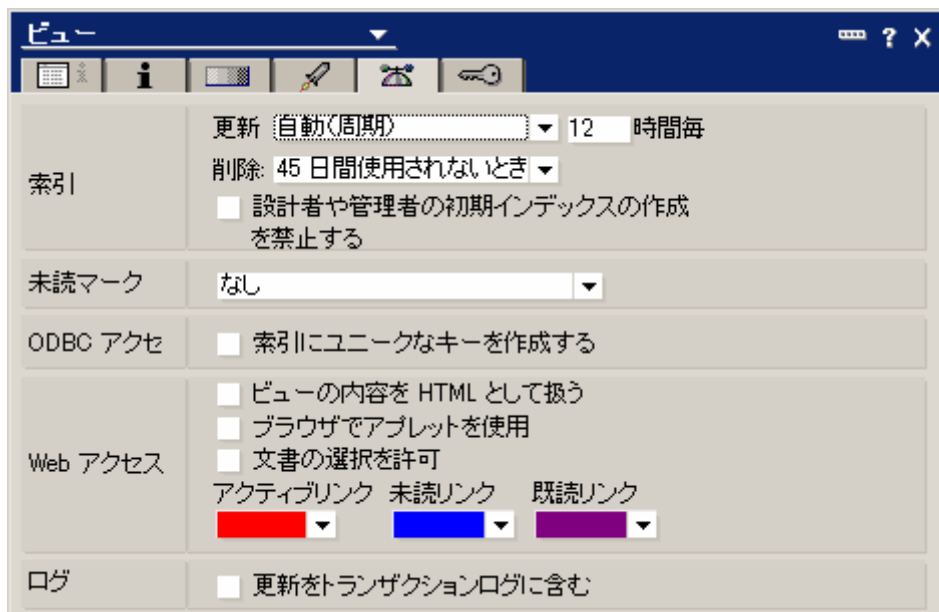
「ビューのプロパティ」ダイアログ・ボックスには、ビューの索引作成を制御するオプションのセットが含まれています。これらのオプションはほとんど使用されませんが、適切な索引作成オプションを選択することで、パフォーマンスが大きく向上する可能性があります。

たとえば、ユーザーがフォーム上のキーワード・リストに入力をするため、頻繁に検索が発生するような特殊なキーワード文書が含まれるデータベースがあるものとします。キーワード文書はほとんど変更されませんが、アプリケーション内の他の文書は常に変更されています。

@DbLookup の説明から、このような検索にはキャッシュを使用することが最良であることがわかっていますが、キャッシュ値がない最初の検索ではビューにアクセスしなければなりません。これを行うとき、Lotus Notes はビューの最後の使用後に変更文書があることを認識し、これらの文書をチェックし、ビュー内に存在しないことを検出するために時間を費やします。

キーワード値として@DbLookup で使用するビューは、使用されるたびに索引を再作成する必要はありません。このようなビューには、索引作成オプションの「自動(周期)」を選択し、「x 時間毎」の x に適切な値を設定するとよいでしょう(図 2 参照)。

図 2. ビューの索引作成オプション



これらのビューは、ユーザーがまったく使用しない場合でも、頻繁ではありませんがサーバーによって更新されます。時々、不運なユーザーは、索引の再作成に遭遇することがあります。しかし、平均検索時間はかなり短くなり、複雑なフォームで検索する各ユーザーがこのコストを負担することがなくなるため、このオプションを使用しないときよりもフォームは速く開きます。

四半期のレビュー用に、ビューを四半期ごとに1日使用する場合は、索引を45日間保持する意味がありません。2日後に索引を破棄するよう設定すると、サーバーの作業を削減できます。

適切な索引作成オプションを選択することでパフォーマンスを改善できる状況は他にもあります。ビューごとに適切な設定を決めることが重要です。

メモ:NotesView.Refresh メソッドを使用し、現在のレプリカでプログラマチックに索引を更新することは可能です。たとえば、通常はほとんど更新されない索引があり、ビューにデータを表示する特定の文書を保存する際に、ビューを更新して、検索で新しいデータをすぐに使用可能にしなければならないケースを考えましょう。このような場合は、フォームの Postsave イベントで、Refresh メソッドをビューに対して使用します。同時に、ReCache を指定して@Db 関数を使用し、ビューへの特定の検索のキャッシュを更新します。

6.9 読者フィールド

読者フィールドは、必要であればこれに代わるものはありませんが、ビューのパフォーマンスをかなり遅くする可能性があります。読者フィールドを持つ文書が含まれているビューをユーザーが開くとき、Lotus Notes は行を下方向にスキャンし、ユーザーがアクセス権を持っている文書を検索します。画面に表示するだけの文書が得られると、検索は停止します。1つの文書だけにアクセス権を持つ場合は、ビュー内の各行を検索し、その文書を特定しなければなりません。この処理は時間がかかることがあります。

これに関しては、以下のような対策があります。

- 読者フィールドの値を短くします。アクセス名が含まれる長いリストよりも、1つのロールのメンバーをチェックする方が、速くなります(また、ロールを使用する方が保守容易で有利です)。
- そのようなアプリケーションではビューの使用を避けます。ユーザーが1つまたは2つの文書にしかアクセスできない場合は、たとえば、それらの文書へのリンクを含む電子メールを自動的に送信するなど、他の方法を通じたアクセス方法を提供します。
- アクセスできる文書が含まれるカテゴリだけを表示する単一カテゴリの埋め込みビューを使用します。
- 空のカテゴリ(つまり、ユーザーが表示できる文書を持たないカテゴリ)を表示するように設定されたカテゴリ化されたビューを使用します。もちろん、ユーザーを文書に導く手段を提供しない限り、ユーザーは自分自身の文書を見つけるのが困難であるため、この方法は、ユーザー用のカテゴリだけを表示する@SetViewInfo とともに使用するとよいでしょう。

メモ:カテゴリ化されたビューを使用する方法には、セキュリティーへの影響があります。つまり、他の方法ではアクセスできない文書の1フィールド(カテゴリ)がユーザーに表示されることとなります。この点について了解を得る必要があります。

- ユーザーにローカル・レプリカの使用を推奨します。ローカル・レプリカにはユーザーがアクセス権を持つ文書だけが含まれるので、表示できない文書を除外する余分な作業は不要です。

読者フィールドは、ナビゲーションの補助のみのために使用しないでください。たとえば、表示可能な文書だけをビューに表示し、ユーザーが文書を見つけやすくする手段として用

いるケースです。文書内の情報が実際には機密でない場合は、上記または次のセクションで説明するように、より良い方法があります。

6.10 最初は個人共有ビュー

@UserName および@UserRoles は、共有ビューの選択式または列式で使ったときに、望ましい結果を生成しません。これは、ユーザー個人の文書だけを表示するために開発者が「最初は個人」ビューを作成する最も一般的な理由です。これらのビューは、サーバーに保管されるか(通常、アプリケーションのパフォーマンスに影響します)、またはユーザーのローカル「デスクトップ」ファイルに保管されます。

デスクトップ・ビューはサーバーのパフォーマンスに直接影響しませんが、他のビューと同様に、ユーザーが開くと、最新の変更を表示するために索引が再作成されます。このことは、ユーザーのクライアントが、ビューの最後の使用以降に変更されたすべての文書をサーバーに要求しなければならないことを意味します。これはユーザーの待ち時間につながり、多くのユーザーがこれを行うと、「すべてのデータの送信」を求める多数の要求により、サーバーがダウンする可能性があります。

ビューの索引作成は要約データのみを使用するため、大きいリッチテキスト・フィールドおよび添付ファイルはここでは問題となりません。

パフォーマンスの問題に加え、開発者にはユーザー個人が持っているコピーの設計を更新する方法がないため、個人ビューはメンテナンスの課題も抱えています。共有列もこのコンテキストでは機能しません。ビュー内で共有列を更新するには、更新するユーザーに、そのビューへのアクセス権が必要だからです。

Lotus Notes ビューの「単一カテゴリー」機能を使用して、「最初は個人」ビューを避けることができます。たとえば、ユーザーが自分自身の文書を表示するには、所有者でカテゴリー化するビューを使用し、「単一カテゴリー」の式とともにそのビューをフォームまたはページに埋め込むか、ビューの **Postopen** イベントで@SetViewInfo を使用して現在のユーザーに表示を制限します。共有ビューは 1 つだけなので、索引作成のコストは全体的に最小限となります。また、索引は常に最新に近い状態なので、個々のユーザーは、デスクトップの個人ビューのように待たされることはありません。

7 コード

LotusScript コードまたは **Java™**コードを書き始めると、パフォーマンスを低下させる機

会を持った新たな世界が出現します。ここでは、一般的な落とし穴について説明しましょう。

7.1 GetNthDocument

`NotesDocumentCollection.GetNthDocument` を使用してコレクションを反復アクセスするのは非常に低速です。代わりに、`GetFirstDocument` および `GetNextDocument` を使用してください。`GetNthDocument` を効果的に使用できるコレクションのタイプもありますが、使用しない方が簡単です。

7.2 フォームまたはビューのアクション・コードが多すぎる

フォーム、ビュー、またはフォルダにたくさんのアクションがあり、設計要素の各アクションにコードを記述すると(または、共有アクションを使用する場合でも)、設計要素を使用するたびにメモリーにロードしなければならない多数のコードを格納することになります。

ほとんどの場合、多くのアクションのうち 1 つまたは 2 つのアクションだけが使用されるので、すべてをロードする時間は無駄になります。アクションが複数の場所で使用されていると、設計キャッシュ内で同じコードを複数回キャッシュすることになり、他の目的で必要になるかもしれないメモリーを消費してしまいます。

いくつかのアクション・コードをエージェントに移動することを考慮してください。これにより、ユーザーが実行を要求したときにのみメモリーにロードされるコードの唯一のコピーを持つことになります。アクション・ボタンは、`@Command([RunAgent])` を使用してエージェントを呼び出すようマクロ言語で書くことができるため、設計要素とともに非常に少ないコードがロードされます。

これは、ユーザーに個人ビューまたは個人フォルダの作成を認めるときに、特に重要です。アクション・コードは、ユーザーのフォルダに何回も複製されてスペースを占有し、ユーザーが手動で個人ビューを削除するまで更新できないからです。

7.3 スクリプト・ライブラリーが多すぎる

同じスクリプトで複数のスクリプト・ライブラリーをロードするのに要する時間は、リニアな関係を超えます。つまり、特にライブラリーが他のライブラリーを使用している場合、10 個のスクリプト・ライブラリーをロードするのに、5 個のスクリプト・ライブラリーの 2 倍を超える時間がかかります。

将来、変わる可能性があります、分岐点は存在します。2つの設計要素にアクセスする時間は、その合計と同じ量のコードを持つ1つの設計要素にアクセスするよりも、多くの時間がかかります。特定のエージェントで呼び出さないコードをインクルードする場合があっても、使用頻度の高いスクリプト・ライブラリーをまとめることにより、ロード時間を節約できます。

7.4 ComputeWithForm

NotesDocument の ComputeWithForm メソッドは、コードを書かずに文書内の計算結果フィールドの値を更新する簡単な方法です。残念ながら、手動で計算して新しいフィールド値を代入するよりもかなり遅くなります。ComputeWithForm を使用しているエージェントが遅い場合、この呼び出しを取り除き、特定のフィールドに代入する数行のコードを挿入することで、速度を大幅に向上できる可能性があります。

7.5 自動更新ビュー

NotesView オブジェクトを使用するときは、デフォルトで、ビューの標準の索引更新属性がインプリメントされます。たとえば、「野菜」文書のコレクションを更新するケースを考えます。処理の一部で、同じデータベースの「害虫」ビューで野菜の害虫を検索する必要があります。しかし、野菜文書を保存するときに、その時点で文書が更新されています。

次の文書に進み「害虫」ビューを検索したときに、Lotus Notes はビューの索引が古いことに気づき、索引を更新します。ユーザーは変更内容が「害虫」ビューに影響を与えないことを知っていますが、Lotus Notes は変更された文書をテストするまではそのことを知りません。

このようなケースでは、Lotus Notes ビューの「自動更新」プロパティを使用し、ユーザーが Refresh メソッドを使用して明示的に更新を要求しない限り、ビューの索引を更新しないように Lotus Notes に指示するとよいでしょう。これにより、大幅にスピードが変わります。

ユーザーの変更が NotesView の内容に影響を与える場合でも、実行する内容に対してそれが問題とならないことを認識していれば、この方法を使用できます。たとえば、更新によってビューから文書が削除されることを認識していても、次の文書に移動しているので、それは問題になりません。

7.6 効率のよいコレクション・ベースのメソッドの使用に失敗

NotesDocumentCollection クラスには、名前が「All」で終わるいくつかのメソッドがあり

ます。これらのメソッドを使用すると、コレクション内のすべての文書に処理を実行できます。コレクションを反復アクセスして各文書を個々に操作するよりも、これらのメソッドを使用する方がかなり速いので、これらのメソッドに習熟するとよいでしょう。（もちろん、各文書に複数の処理が必要でない限りです。このような場合は、反復アクセスする方が、各文書を1回だけ保存するので速くなります。）

7.7 高負荷のオペレーションの繰り返し

組み込みクラスには、きわめて遅いメソッドとプロパティーがいくつかあります。不要な場合に、これらの関数を繰り返し使用することを避けると、コードをより速く実行できます。たとえば、文書のコレクションを処理するケースを考えます。各文書でいずれか1つのフィールドを検索値として使用し、他のビューから情報を取得します。

```
Dim view As NotesView
```

```
Set docCur = coll.GetFirstDocument
```

```
Do Until docCur Is Nothing
```

```
Set view = db.GetView("CustomersByID") ' oops! Don't do this in the loop! Set docCust =  
view.GetDocumentByKey(docCur.CustID(0), True)
```

```
Loop
```

```
Set docCur = coll.GetNextDocument(docCur)
```

このサンプル・コードでは、コレクションに1000文書が含まれる場合は、高負荷な `GetView` メソッドを1000回も呼び出します。`GetView` が1回だけ呼び出されるように `Do Until` 行と `Set view` 行の位置を交換すると、コードはかなり速くなります。

エージェント・プロファイラーは、このようなことを検出するのに役立ちます。エージェント・プロファイラーの詳細については、[developerWorks Lotus](#) の記事「[アプリケーション・パフォーマンスのトラブルシューティング:パート1:トラブルシューティングの手法とコードのヒント](#)」および「[アプリケーション・パフォーマンスのトラブルシューティング:パート2:Lotus Notes/Domino 7の新規ツール](#)」を参照してください。

7.8 変更されていない文書の保存

パフォーマンスに影響する要因の1つに「変動」、つまり文書が変更される頻度があることを思い出してください。文書処理するエージェントを書くときには、不必要に変更を文書に保存しないようにします。アイテムに値を代入する前に、アイテムがすでにその値を

持っていないかチェックします。何も変更せずに終わった場合は、**Save** メソッドを呼び出さないでください。検索メソッドを使用し、処理する必要のない文書をコレクションから除外できることがあります。

アイテムを定期的にチェックして変更が必要かどうか判断している場合、エージェントの実行時間は少し長くなるかもしれません。あるいは、メモリー内の情報を比較するよりも文書を保存する方が大幅に時間を要するので、そうならないこともあります。どのケースでも、アプリケーションの他の部分は、複製からビューの索引作成、全文索引作成まで、より効率的に機能します。

不要な保存を避けると、複製競合の可能性も低下します。複製はアイテムの変更時刻を使用し、文書全体を他のレプリカには送信せず、変更されたアイテムだけを送信します。このため、文書を保存しなければならないのであれば、新しい値が必要なアイテムだけを変更することで、複製の時間を削減できます。ローカル・レプリカを使用するユーザーは喜ぶでしょう。

7.9 文書の検索方法

ほとんどのエージェントが実行しなければならない作業の 1 つに、処理する文書セットの特定があります。これを行うにはさまざまな方法があり、それぞれ異なる状況に適しています。

文書のコレクションを検索および処理するさまざまな方法については、**developerWorks Lotus** の記事「[Lotus Notes/Domino 7 アプリケーションのパフォーマンス:第 1 部:データベース・プロパティと文書コレクション](#)」で解説されています。まとめると、次のようになります。

- 目的の文書を含むビューがあり、有用な方法でソートされている場合は、**GetAllDocumentsByKey** メソッドを使用してビューからその文書を読み込むことが、通常最も速い方法です。
- 大量の文書があるデータベースでは、データベースの全文索引が作成されている場合、**FTSearch** メソッドは **NotesDatabase.Search** よりも高速です。全文検索は、エージェントの「文書の選択」イベントに入れることによっても実行できます。

どちらのケースでも、サーバーによってあらかじめ実行されていた索引作成処理を利用して時間を削減できます。このため、各文書を個々にテストする必要がある

NotesDatabase.Search と比較して、実行時に行う処理が少なくなります。

全文検索では、NotesDatabase.Search が行うように詳細なレベルまで文書を除外することはできませんが、通常、検索結果に反復アクセスし、適用されない一部の文書をスキップするだけの時間を節約できます。全文検索の構文の詳細については、Lotus Notes クライアント・ヘルプ (Lotus Domino Designer ヘルプではありません) の「演算子を使用して検索条件を絞り込む」を参照してください。思っていたよりも、かなり多くのことができることに気付くでしょう。

メモ:要件によっては、式ベースの選択条件が設定されているビューで全文検索を実行することにより、式による検索のパワーと全文検索のパフォーマンスを結合できる場合があります。

7.10 使用されていない文書をキャッシュから削除

前のバージョンの Lotus Notes では、Delete ステートメントを使用し、使用済みの NotesDocument オブジェクトをメモリーから削除して、メモリー使用量を改善できました。しかし、バージョン 6.0 以降では、これを行う価値がなくなりました。(これが何を意味するかわかる場合は、あえて実行する必要はありません。意味がわからない場合は、気にすることはありません。)

パフォーマンス以外の理由で、まだ Delete を使用することもあります。たとえば、最後に文書を開いた後、別のプロセスによって文書が変更された可能性があり、最新のデータを確実に読み込みたい場合などです。

7.11 より効率の高いループ、代入など

“for”ループと“while”ループのタイミングやグローバル変数とスタック変数の比較などを記述した資料もあります。しかし、特にアプリケーションが計算主体ではない限り、これらのことからパフォーマンスが大幅に改善される可能性は低いでしょう。

多くのスクリプトでは、変数の値を操作するよりも、文書およびビューを開く方がはるかに多くの時間を費やします。不要な配列参照を避けてミリ秒単位で節約できるとしても、不要なビューを開くことは秒単位の時間がかかります。パフォーマンスの改善に時間をかける場合は、大きな見返りを得られるアイテムから始めましょう。

LotusScript のさまざまな式およびステートメントのパフォーマンス特性を知ることは役立ちますが、オリジナルのコードを記述するときに良い習慣を構築する方がより価値があ

ります。後で見直し、非効率的な代入を修正しても、得られるものほとんどありません。

この領域で価値のあるヒントは以下のとおりです。

- `GetNthDocument` は使用しないでください(前述)。
- デフォルトの `Variant` 型を使用せず、変数を明示的に宣言してください。これにより、より良いパフォーマンスを得られるだけでなく、コンパイル時にすぐにエラーを見つけるのに役立ちます。プログラミング・ペインのプロパティで、`Option Declare` ステートメントを自動的に `LotusScript` コードに挿入するオプションを使用します。

7.12 LC LSX の使用

外部のリレーショナル・データベースまたはデータ・ファイルを統合する場合は、組み込みの ODBC クラスよりも、LC LSX API の方が一般的に速くなります。IBM Redbooks 資料「[Implementing IBM Lotus Enterprise Integrator 6](#)」(US)には、この API を使用してプログラミングする方法やそのパフォーマンスを最大限に高める方法に関する数多くの情報が記載されています。

8 テスト

本書の最初の部分で述べたように、データ量やユーザー数が少ないときには良好に機能した小規模アプリケーションが、データとユーザーが増えるにつれて機能不全に陥ることが数多くあります。設計をテストするときは、大量の文書を使用したり、50 人のユーザーが同時に使用するとどうなるかを確認めたりするとよいでしょう(時間に余裕のある 50 人のユーザーをそろえられない場合は、この状況をシミュレートする自動化されたテスト・ツールを使用できます)。

また、少量のサンプル・データのセットを用いて、選択したフィールドにランダムな値を入力し、それを何倍にもして数千文書へと増加させるエージェントを書くのは難しくありません。また、新規文書を作成するエージェント・オプション(エージェント編集画面の右下の部分にあります)を使用する場合は、式エージェントで同じことができます。

ただし、次の点に注意してください。すでに稼働しているアプリケーションをテストするときは、非実動サーバー上のデータベースのコピー(レプリカではありません)でテストを行います。また、できれば実動サーバーと複製しないサーバーを選択します。このようにすると、実動データを破損するリスクを負うことなく、ユーザーが作業に使用しているサーバーをクラッシュまたはダウンさせることもありません。

9 プロフィール文書の使用

プロフィール文書は、あまり頻繁に変更されない情報を格納および取得するのに適した方法です。最初の使用時に文書全体がキャッシュされるので、カスタマイズ可能なキーワード・リストをここに保管すると非常に効率的です。ビューの索引作成に問題はなく、キャッシュの制御について心配する必要もありません。プロフィール文書は通常の文書と同じように複製されます(複製選択式を設定しているユーザーは、プロフィール文書を除くことによりアプリケーションを誤って壊すことがないため、この点でも通常のキーワード文書よりも優れています)。できるだけ使用してみましょう。簡単かつ興味深い使い方ができます。

10 まとめ

長くなりましたが、このホワイト・ペーパーは、すべてを網羅しているわけではありません。役に立つ追加情報やテクニックについては、以下の「リソース」セクションを参照してください。また、Lotus 指向のさまざまなブログや Wiki の閲覧を続けることも役立ちます。パフォーマンスに関連するヒントが頻繁に書き込まれています。

11 リソース

- IBM Redbooks 資料「[Domino アプリケーションのパフォーマンスに関する考慮事項](#)」(2000年3月)は、パフォーマンス問題に関する標準的なリソースです。やや古いですが、十分役に立つ内容です。
- developerWorks Lotus の記事「[Lotus Notes/Domino 7 アプリケーションのパフォーマンス:第1部:データベース・プロパティと文書コレクション](#)」では、文書のコレクションを検索および処理するさまざまな方法が解説されています。
- developerWorks Lotus の記事「[Lotus Notes/Domino 7 アプリケーションのパフォーマンス:第2部:データベースのビューの最適化](#)」では、さまざまな種類のビューの索引作成時間を調べ、読者フィールドのパフォーマンスへの影響を削減する方法が解説されています。
- developerWorks Lotus の記事「[アプリケーション・パフォーマンスのトラブルシューティング:パート 1:トラブルシューティングの手法とコードのヒント](#)」および「[パート 2:Lotus Notes/Domino 7 の新規ツール](#)」では、スピード低下の原因となるアプ

リケーションの部分を特定する方法が解説されています。

- developerWorks Lotus の記事「[アプリケーションのパフォーマンス・チューニング 第1回](#)」では、パフォーマンスに影響を与えるデータベース・プロパティが詳述され、ビューの索引作成時間のログを記録する方法が解説されています。
- developerWorks Lotus の記事「[アプリケーションのパフォーマンス・チューニング 第2回](#)」では、主に不要な計算を防止してパフォーマンスを向上させる方法が解説されています。
- IBM Support Web サイトの Techdoc 「[Time/Date views in Notes: What are the options?](#)」(US)では、日付/時刻に基づくビューを作成する別の方法が解説されています。
- IBM Business Partner の文書「[Performance Engineering Notes/Domino Applications](#)」(US)では、本書と同様の内容の一部を取り上げ、文書とフィールドの増加による効果が正確に比較測定されています。Web アプリケーション専用のヒントも含まれています。
- [この Lotus Sandbox のダウンロード](#)(US)には、Replicate Customers と呼ばれるエージェントが含まれています。このエージェントは、各文書を削除および再作成せずに、一方向の同期を行うアルゴリズムを示します。
- developerWorks Lotus の記事「[複製アプリケーションで順序番号を生成する](#)」では、複数のレプリカにわたって連続番号を取得する 1 つの方法が解説されています。

12 著者について

Andre Guirard は Lotus Domino Designer の開発チームのメンバーです。長年 Lotus Notes 開発者として活躍し、現在は開発者の育成に重点を置いています。developerWorks、The View、および他の出版向けの多数の技術記事を執筆し、IBM Redbooks の投稿者でもあります。また、developerWorks blog のブログ「[Best Practice Makes Perfect](#)」を運営しています。

Lotusphere および IBM コンファレンスで講演することもあります。休日には、奥さんのガーデニング・プロジェクトを手伝い、SF やファンタジー小説を書いています。

13 謝辞

著者は、John Curtis 氏およびこのホワイト・ペーパーの専門的なレビューをお手伝いいただいたビジネス・パートナーの方々に謝意を表します。

商標

- IBM、DB2、Domino、Lotus および Notes は、International Business Machines Corporation の米国およびその他の国における商標です。
- Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。
- 他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

IBM の著作権および商標の情報:<http://www.ibm.com/legal/copytrade.phtml>