多層分散型基幹業務システム構築の課題と解決

ＮＵシステムズ株式会社
TEL:06-6233-8200

内容

【システム説明】
今回のシステムの導入に至った経緯
次期システムの基本要件
システム構築スケジュールと開発規模
3層構造、DataSnap採用に至った経緯
全体システム概要とシステム構成の説明
DataSnap3層構造の全体システム説明図

【DataSnapでの実装】
DataSnapを使った二層ライクな開発
多接続環境でのパフォーマンスの維持
カーソルセットの取扱いで注意すべきこと
DataSnapで発生する、ＳＱＬステートメント20k問題
DataSnapのDelta処理でのBlobデータの取扱い
課題とＴＩＰＳ

今回のシステムの導入に至った経緯

基幹システムでありながら、Webベースで開発・運用されている（2013/1）

消費税率の変更に対応していない

Webベースの為、セキュリティが確保しきれない

Web標準やHTML5など、ブラウザ依存でシステム側がフォローし切れない

ユーザー接続数、仕様の追加が頻繁に発生する

モバイル環境の増大

消費税率の変更まで、１年しかない

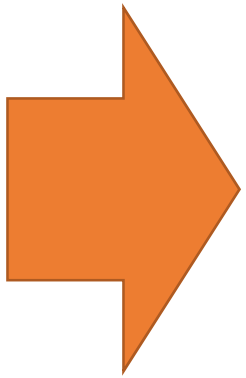
全く新しいシステムが必要（2013/4）

次期システムの基本要件

ユーザー規模（システム運用開始時点）

業種：フランチャイズチェーン店
現有店数１００店舗
同時接続数５００

１８時間３６５日運用
固定接続環境



システム設計要件

現有店数３００店舗
同時接続数１５００

２４時間３６５日運用
モバイル環境、
固定接続環境併存

１０年間稼働保証
セキュリティ確保

柔軟なシステム変更
短期間でのシステム構築

システム構築スケジュールと開発規模

２０１３年１月～３月	要件定義開始
２０１３年４月	基本開発契約締結
２０１３年４月～６月	システム要件定義、要素技術開発
２０１３年６月～９月	プロトタイプ版開発
２０１３年10月～２月末	本システム開発
２０１４年１月～３月末	ユーザーテスト、データー移行
２０１４年４月１日	本稼働

総開発ページ数	フロントエンド	４００ページ
	外部連携サーバ	３サイト
	サーバサイドモジュール数	３０本

3層構造、DataSnap採用に至った経緯

短納期で要件定義からプロトタイプ作成、デバッグ、納入支援まで
少人数で完成させる必要

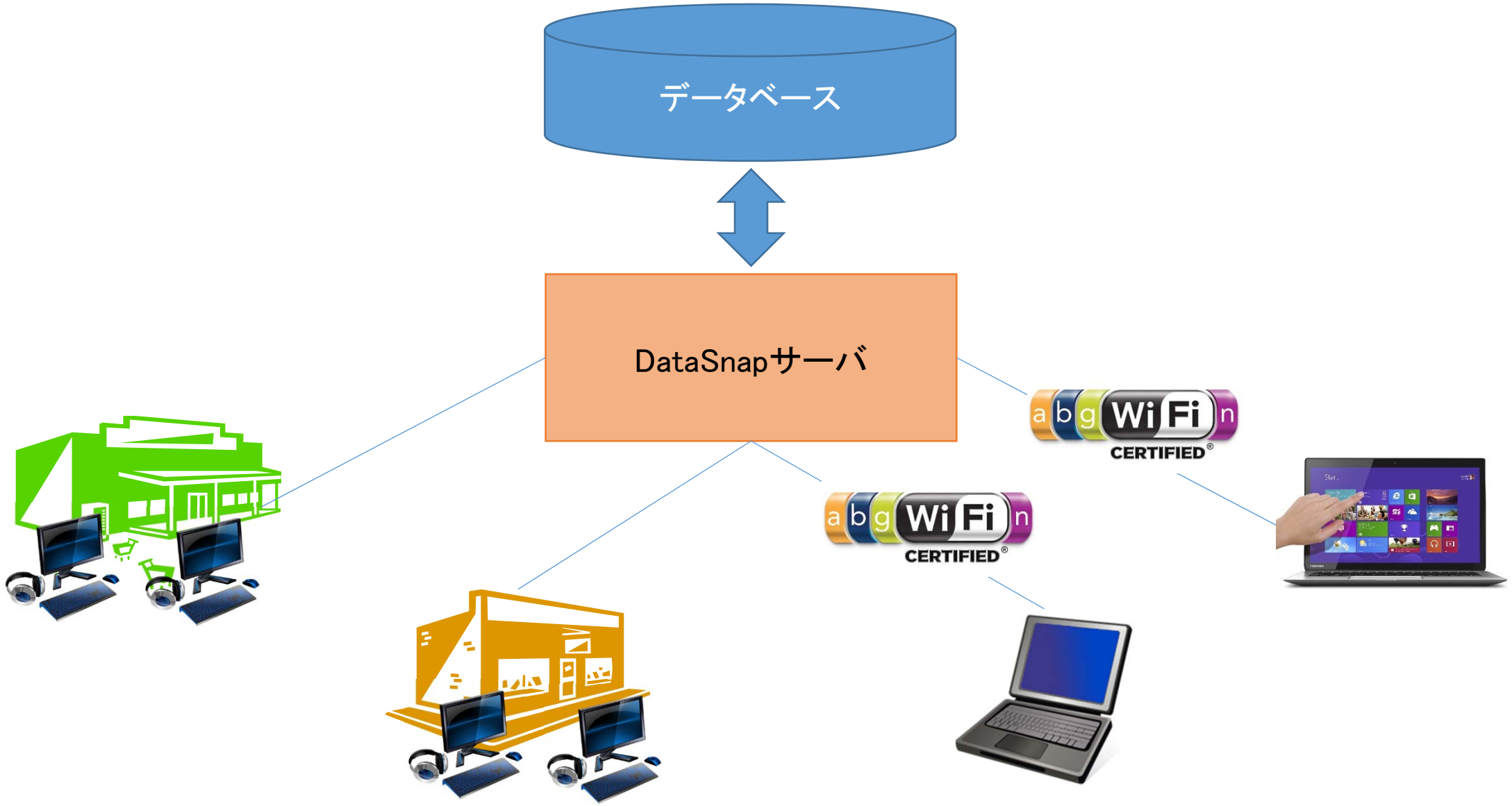
長期間に渡り安定稼働を保証し、セキュリティを確保した上で
ユーザー接続環境に柔軟に対応できること

ユーザー環境に左右されない長期間のシステム運用の保証

ユーザーの運用要件で多数発生する、複雑な業務処理を
リアルタイム処理する必要

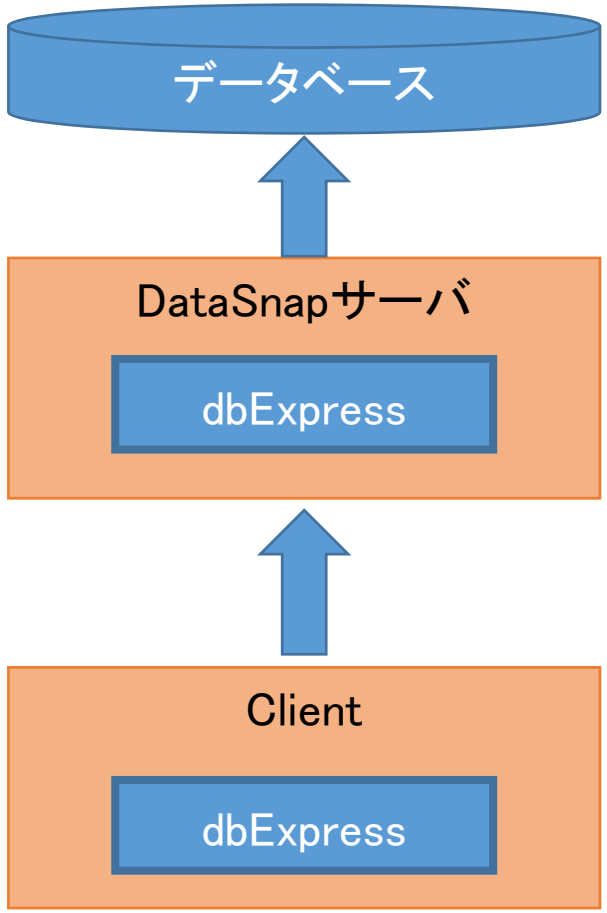
Webベースではない為、クライアントのプロファイル情報を確実に取得
最新のWindowsプラットフォーム上で運用可能なこと⇒Windows8対応
サーバへの負荷を低減する為、フロントエンドで殆どの処理を完了したい
システムの停止を行うことなく拡張が行える、スケーラビリティの確保

DataSnap3層構造の全体システム説明図



DataSnapを使った二層ライクな開発

当初の構成予定



dbExpress一本での実装を検討。

⇒FireDACのリリース。

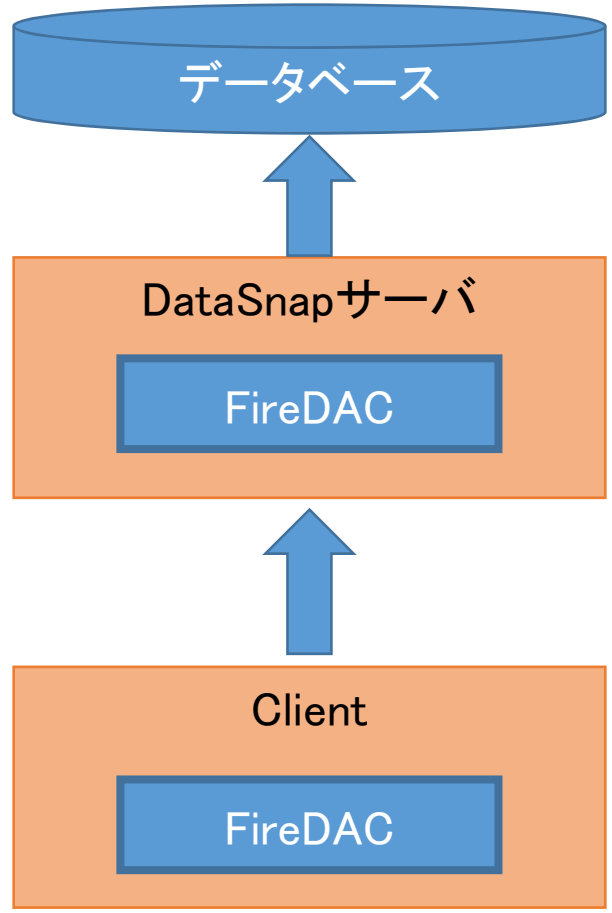
比較した結果

・フェッチ件数制御：
　FireDACなら意識する必要なし

・フェッチ速度：
　FireDACが圧倒的に高速
　(dbExpressでも実装は可能だが
　処理が複雑化)

・制御の簡易化：
　FireDACなら、最小3個程度の
　コンポーネントで実装可

改善案

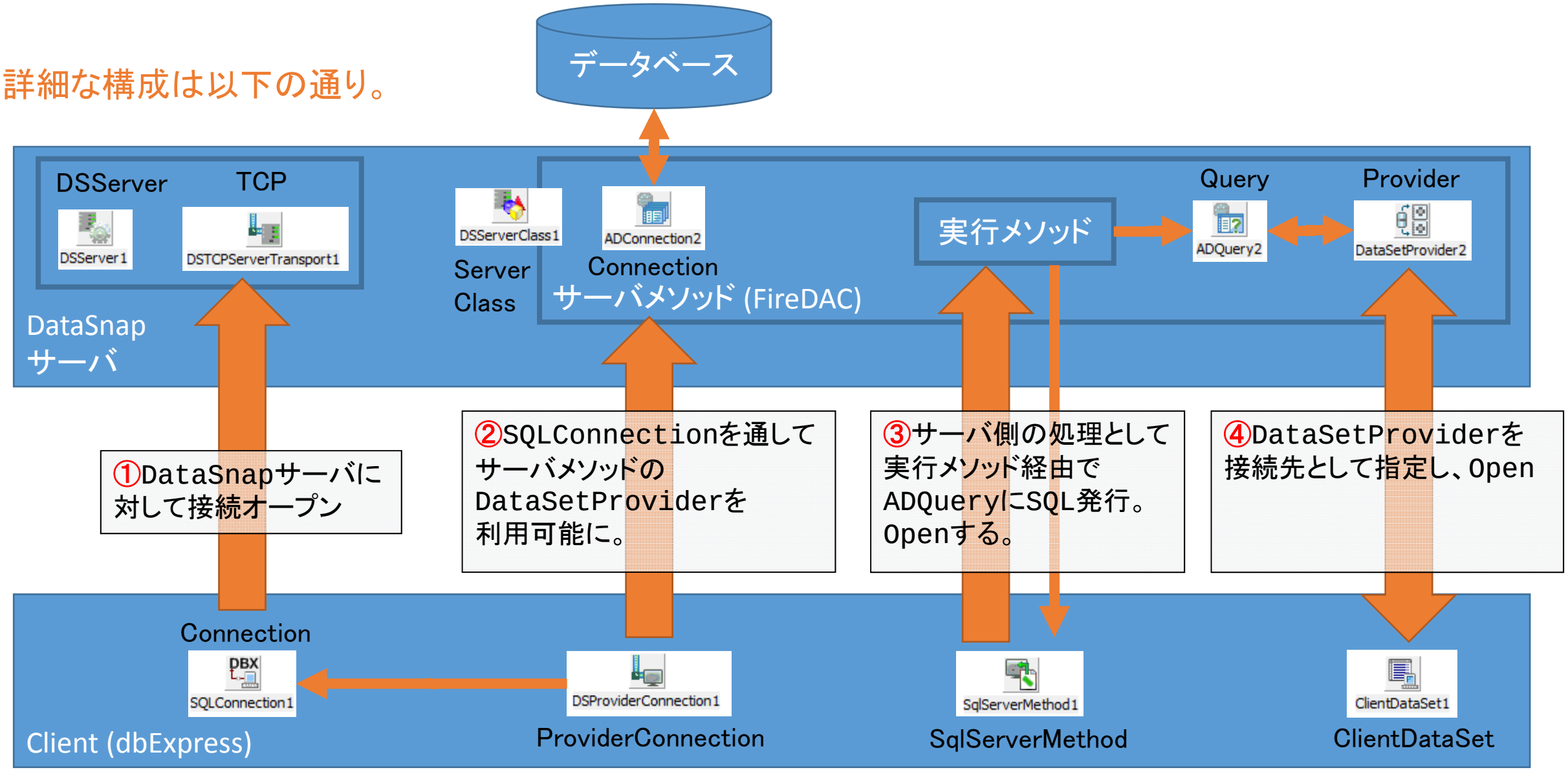


DataSnapを使った二層ライクな開発

しかし問題発生

DataSnapを使った二層ライクな開発

詳細な構成は以下の通り。



DataSnapを使った二層ライクな開発

双方向カーソルでの注意点

●行移動の際、DataSnapサーバに対して通信が発生。（カーソル位置の通知）
⇒行移動が多発する画面、処理では単方向カーソルの方がパフォーマンスが良いケースも。
⇒単方向カーソルの場合、SQLによる更新が必要となる。

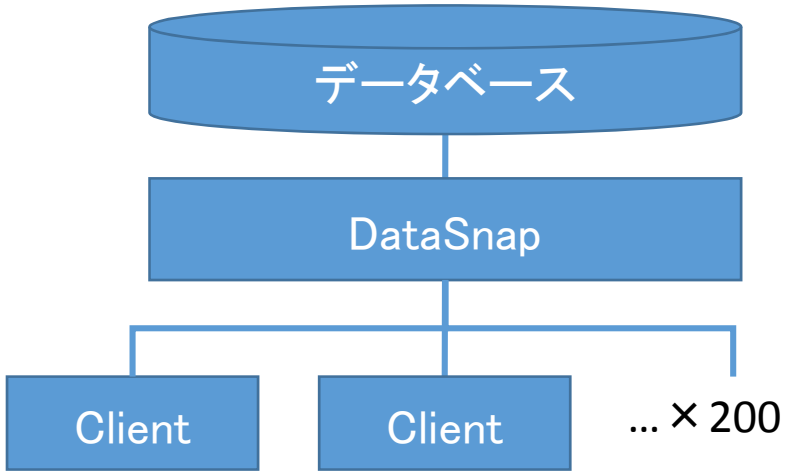
●二層での双方向カーソルデータセット同様、テーブル結合したクエリには使用不可。
⇒極力テーブル結合が発生しないテーブル設計が非常に重要。（正規化を落とすことも重要）

●ClientDataSetのProviderName指定を誤ると、最悪想定したテーブルとは別のテーブルが更新される。
⇒DataSnapサーバ側のADQuery、DataSetProviderの組み合わせは、同じ数字としておく。

●トランザクション制御は、DataSnap側のデータベース接続（=ADConnection）で行う必要がある。
クライアント側の接続(=SQLConnection）のトランザクションではない。
⇒DataSnap側にADConnectionに対してStartTransaction、Commit、Rollbackを実行するだけの
サーバメソッドを用意。

●IDEでのデザインでは、単方向カーソルのみが使用可能。
⇒フィールドの取得などで使用するための、本処理とは別のSQLServerMethod、DataSetProvider、
DataSourceが必要。

多接続環境でのパフォーマンスの維持

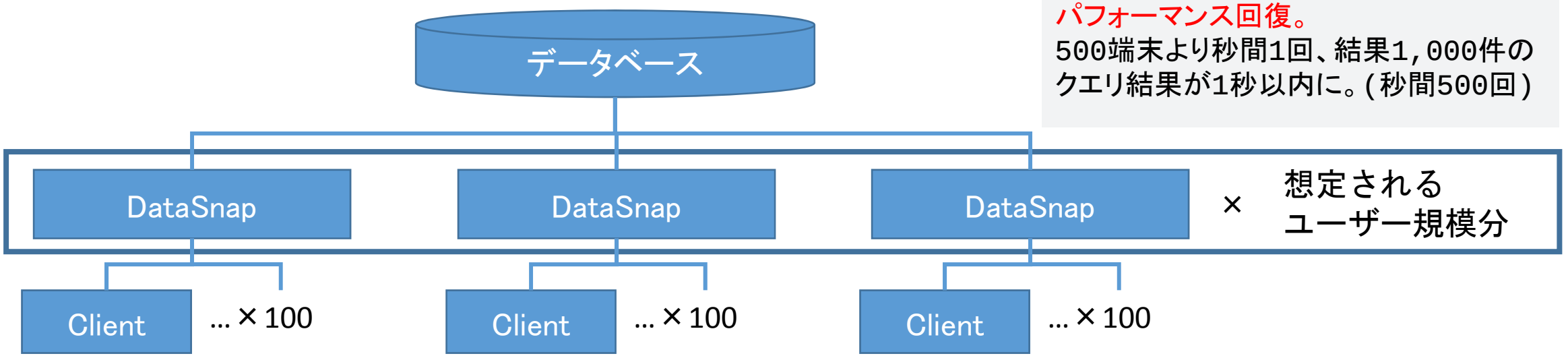


・1000件程度のselect結果が返るまで、時間が30秒近くに…
・150端末前後から、DataSnapに接続すらできない…

■サーバの接続制限？ ⇒ 無制限の状態
■DBサーバの負荷？ ⇒ ほぼ横ばい
■DataSnapの制限？ ⇒ 多くのプロパティがあるが、情報が少なく
全パターン試行に…

大きな工数増大の危機…

●一つのDataSnapでダメなら、複数で処理すればいいのでは？

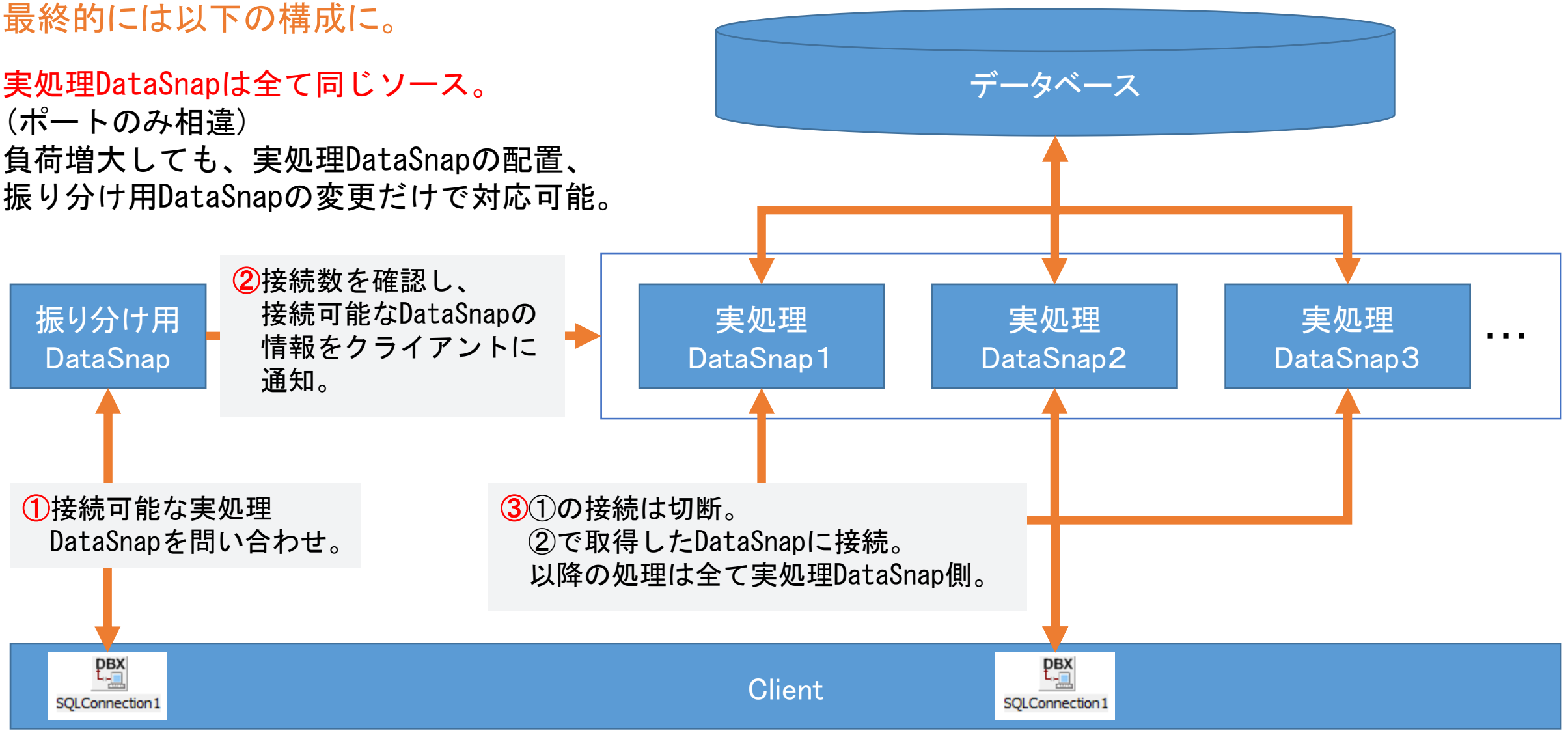


パフォーマンス回復。
500端末より秒間1回、結果1,000件の
クエリ結果が1秒以内に。(秒間500回)

多接続環境でのパフォーマンスの劣化を防ぐ

最終的には以下の構成に。

実処理DataSnapは全て同じソース。
(ポートのみ相違)
負荷増大しても、実処理DataSnapの配置、
振り分け用DataSnapの変更だけで対応可能。



多接続環境でのパフォーマンスの劣化を防ぐ

コード例

②接続数を確認し、接続可能なDataSnapの情報を
クライアントに通知。
（振り分けDataSnap側）

```
function TServerMethods1.GetUsePortNo :
integer;
begin
     SqlServerMethod1.ExecuteMethod;
     SqlServerMethod2.ExecuteMethod;

     if SqlServerMethod1.Params[0] <= 101
     then begin
        Result := 10000;
     end
     else
     if SqlServerMethod2.Params[0] <= 101
     then begin
        Result := 20000;
     end;
end;
```

②接続数を確認し、接続可能なDataSnapの情報を
クライアントに通知。
（実処理DataSnap側）

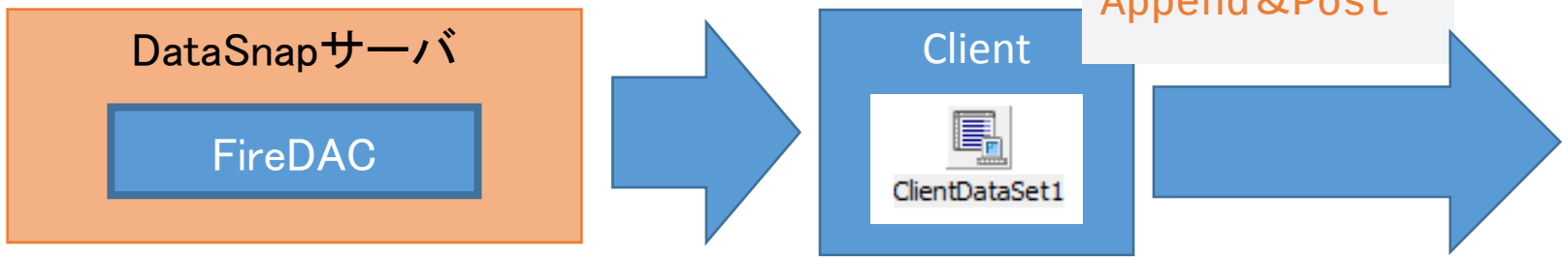
```
function TServerMethods1.SessionCount:
integer;
begin
     Result :=
TDSSessionManager.Instance.GetSessionCount;
end;
```

各実処理DataSnapのポート番号。

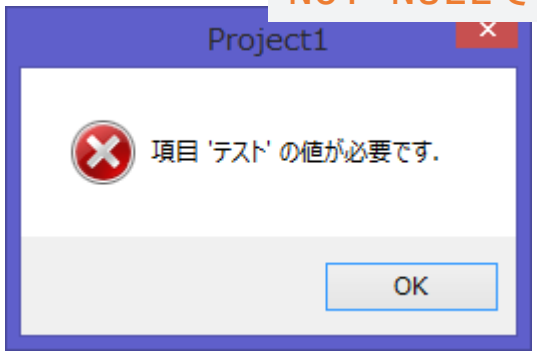
各実処理DataSnapモジュールの
・メモリ使用量
・CPU使用率
などで振り分けることも可能。

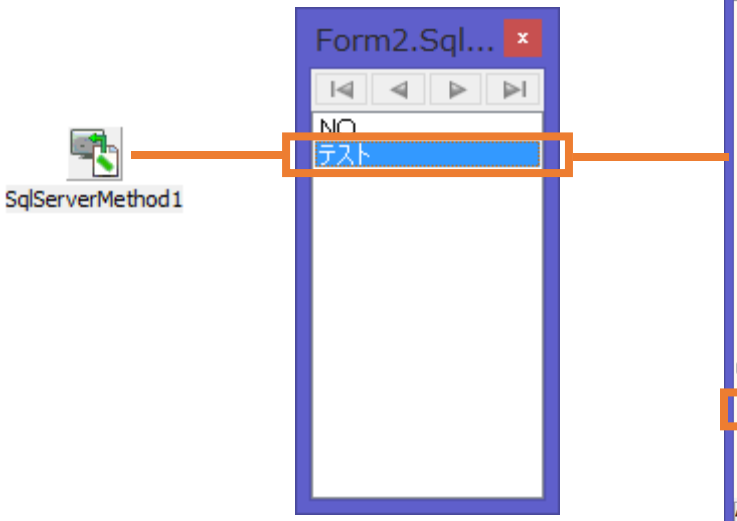
カーソルセットの取扱いで注意すべきこと

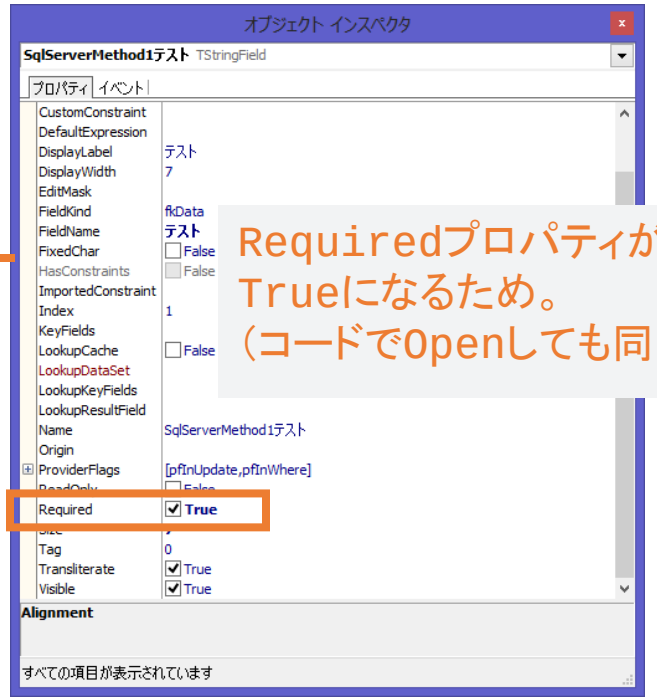
単方向カーソルのデータセットへのデータ追加で、以下の現象。



DBの定義では
NOT NULLでない列…







Requiredプロパティが
Trueになるため。
(コードでOpenしても同じ)

双方向カーソルでは発生しない。

①IDEであらかじめフィールドを
取得しておき、SQLServerMethodの
フィールドの不要なRequiredを外す。

②Open後、コードにてSqlServerMethod
およびClientDataSetの
全フィールドのRequiredを外す。

①の対応がお勧め

DataSnapで発生する、SQLステートメント20k問題

全角文字を含むSQLは20k以上は切り捨てられる…

Select コード1,文字1,コード2,文字2,コード3,文字3,コード4,文字4,コード5,文字5,コード6,文字6,コード7,文字7,コード8,文字8,コード9,文字9,コード10,文字10,コード11,文字11,コード12,文字12,コード13,文字13,コード14,文字14,コード15,文字15,コード16,文字16,コード17,文字17,コード18,文字18,コード19,文字19,コード20,文字20,コード21,文字21,コード22,文字22,コード23,文字23,コード24,文字24,コード25,文字25,コード26,文字26,コード27,文字27,コード28,文字28,コード29,文字29,コード30,文字30,コード31,文字31,コード32,文字32,コード33,文字33,コード34…~~文字34,コード35,文字35,コード36,文字36,コード37,文字37,コード38,文字38,コード39,文字39,コード40,文字40, from テーブル A where コード1 = '001' and コード2 <> 1 …~~

Base64でエンコード、デコードすれば問題なし。

クライアント側

```
SqlServerMethod1.Params[0].AsString :=
                String(EncodeBase64(TEncoding.Unicode.GetBytes(strSQL),
                                    TEncoding.Unicode.GetByteCount(strSQL)));
ClientDataSet1.Open;
```

DataSnap側

```
strSQL := TEncoding.Unicode.GetString(DecodeBase64(AnsiString(strSQL)));
ADQuery1.SQL.Text := strSQL;
ADQuery1.Open;
Result := ADQuery1;
```

DataSnapのDelta処理でのBlobデータの取扱い

双方向カーソルのデータセットのBlob列で、以下の現象。

OldValueとNewValueを
比較しても、バイト数が同じだと
変更があったか判定できない…

```
begin
        by1 := bFld.OldValue;
        by2 := bFld.NewValue;

        if Length(by1) = Length(by2) then begin
            for i := Low(by1) to High(by1) do begin
                if by1[i] <> by2[i] then begin
                   flg := True;
                   Break;
                end;
            end;
         end
end;
```

バイト配列にOldValueとNewValueセット
ループで変更有無をチェック

課題とＴＩＰＳ

【全般】

・実例が少ないため、DataSnap側で細かい制御を行いすぎると、問題発生時の対応に時間が…

⇒DataSnap側は極力シンプルな構成に。
①単方向カーソルのデータセットを返すメソッド
②双方向カーソルのデータセットを開くメソッド
③ＳＱＬを実行するメソッド
④トランザクション制御用のメソッド
⑤排他制御用のメソッド

上記５点があれば、十分システム構築が可能。

【通信】

・クライアント側とDataSnapサーバ側のKeep-Aliveは、デフォルトでは非常に長い(1分)。
通信切断時に応答が帰るまでの時間に直結するため、システム要件に応じた調整が必要。

・DataSnapサーバ側のインスタンス（=DSServerClass)のDestoryイベントで、インスタンス内部の
DB接続を切断する処理を記載しておかなければ、通信切断時もデータベースとの接続が保持されたままに…。