
システムレスポンス改善における手法

富士ソフトABC 株式会社

■ 執筆者Profile ■



引 田 義 仁

1997年 富士ソフトABC株式会社 入社
2003年 6月 現在
IT事業本部 日立事業所 所属
マイコンソフト開発担当

■ 論文要旨 ■

金融システムの開発プロジェクトの支援作業に携わる機会があり、オンライン業務の総合試験の支援作業を行った。この総合試験において項目入力から結果表示、項目入力完了から結果表示・帳票出力までの実行時間が長く、レスポンスがシステム性能として製品レベルに達していないと指摘され、性能対策を行うこととなった。

総合試験の作業フェーズは既に最終調整段階にあり、システムの処理構造を根本から対策することができない状況下で対策を検討して性能改善を行った結果、システム性能を製品レベルとして十分な性能向上を果たすことができた。

そこで、性能向上の結果を得られた性能対策の手順と手段を紹介する。

今回は最終調整段階の作業フェーズの状況下でできる対策ではあるが、今後のシステム開発時の参考になれば幸いである。

■ 論文目次 ■

1. はじめに	《 4》
2. 対策手順	《 6》
3. 調査・分析	《 7》
3. 1 同一処理を繰り返している箇所の洗い出しの結果	
3. 2 不必要な処理（無駄な処理）を行っている箇所の洗い出しの結果	
3. 3 処理を短縮できる箇所の洗い出しの結果	
4. 対策検討	《 8》
4. 1 同一処理を繰り返している箇所の対策検討	
4. 2 不必要な処理（無駄な処理）を行っている箇所の対策検討	
4. 3 処理を短縮できる箇所の対策検討	
5. 改善対策	《 10》
5. 1 同一処理を繰り返している箇所の改善対策	
5. 2 不必要な処理（無駄な処理）を行っている箇所の改善対策	
5. 3 処理を短縮できる箇所の改善対策	
6. 結果（効果）	《 11》
7. むすび	《 12》
7. 1 目標達成について	
7. 2 改善対策の有効性について	
7. 3 今後の展望について	

■ 図表一覧 ■

図 1	単項目チェックの流れ	《 4》
図 2	一括チェックの流れ	《 5》
図 3	DBアクセス	《 10》
表 1	現状と目標の処理時間	《 4》
表 2	単項目チェック処理	《 4》
表 3	一括チェック処理	《 5》
表 4	性能改善対策手順	《 6》
表 5	調査・分析のポイント	《 7》
表 6	初期化処理の重複例	《 8》
表 7	文字列操作関数の例	《 8》
表 8	ログ内容	《 9》
表 9	DBアクセス	《 10》
表 10	対策結果	《 11》

1. はじめに

総合試験中の金融システムのオンライン業務処理において、処理レスポンスが仕様提示レベルに達していないことから、レスポンス改善のための対策を見出すことになった。

現状の処理時間と目標とする処理時間を示す。

表 1 現状と目標の処理時間

	現状の処理時間	目標とする処理時間
単項目チェック	約 2 ～ 5 秒	1 秒以内
一括チェック	約 10 ～ 20 秒	5 秒以内

レスポンス改善が必要な処理の流れは、次の 2 つである。

(1) 単項目チェック (項目入力～結果表示・次項目初期値表示)

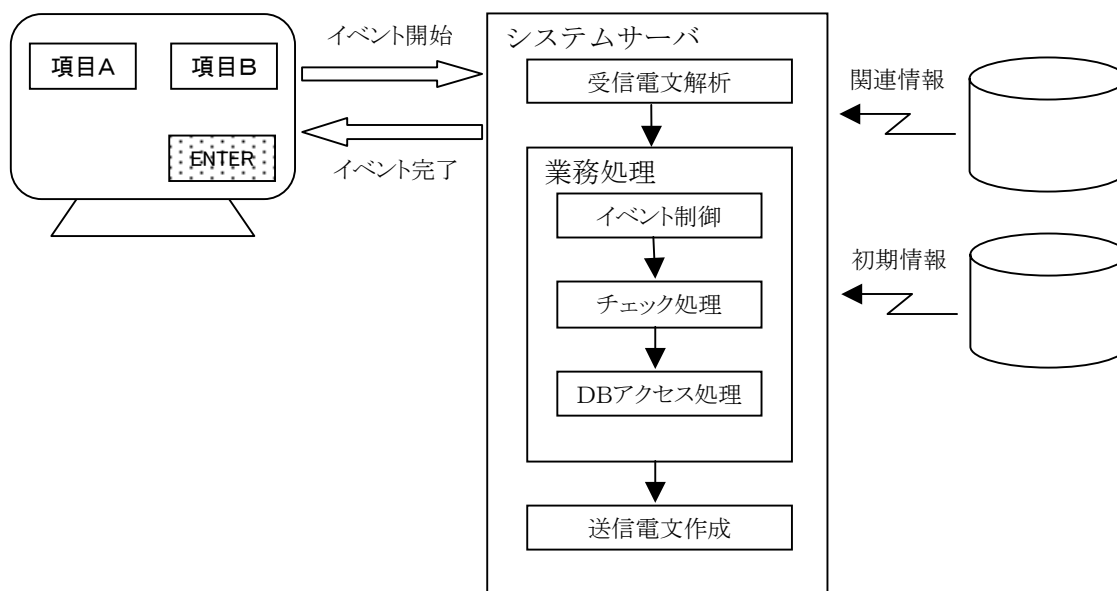


図 1 単項目チェックの流れ

表 2 単項目チェック処理

処理の流れ	処理内容
↓	① 端末から項目入力し、ENTERキー押下 (イベント開始)
	② 端末からシステムサーバに項目情報を通信電文にて送信
	③ 通信電文を解析し、関連する業務処理を起動
	④ 業務処理にて項目チェックに関連する情報をDBから取得
	⑤ 項目チェックし、表示項目を取得
	⑥ 次入力項目がある場合は次入力項目の初期値を取得
	⑦ 表示情報をシステムサーバから端末で通信電文にて受信
	⑧ 端末に表示情報を表示 (イベント完了)

(2) 一括チェック (入力完了キー押下～結果表示・帳票出力)

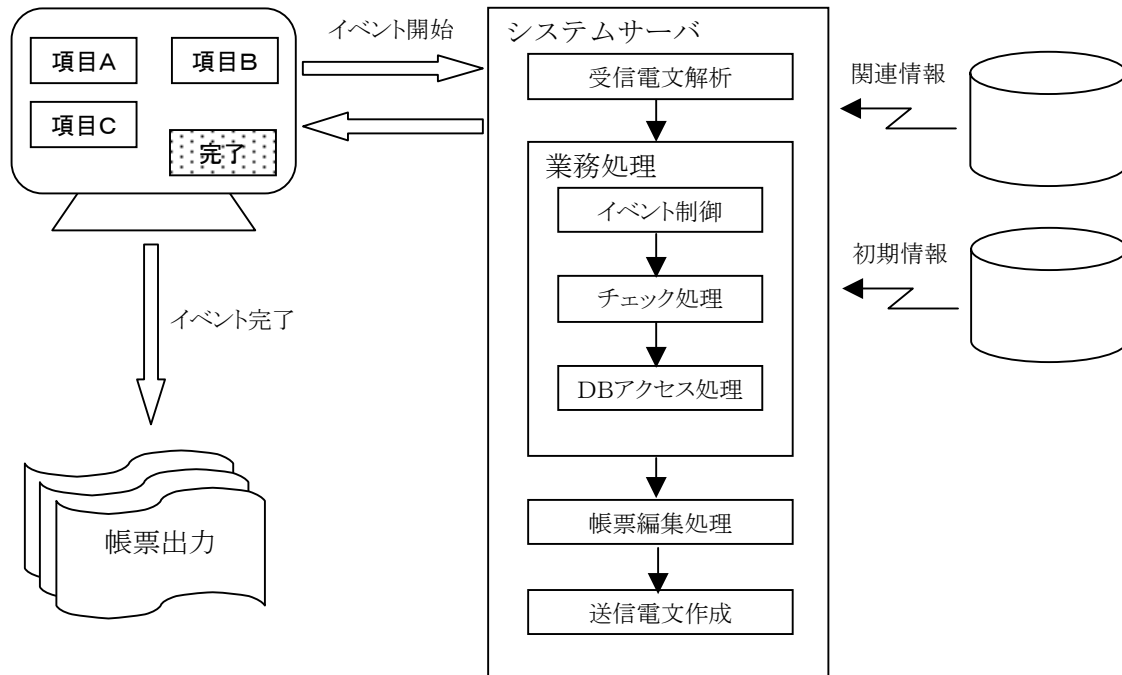


図2 一括チェックの流れ

表3 一括チェック処理


処理の流れ	処理内容
↓	① 端末から全項目入力し，入力完了キー押下 (イベント開始)
	② 端末から全項目情報を通信電文にて送信
	③ 通信電文を解析し，関連する業務処理を起動
	④ 業務処理にて項目チェックに関連する情報をDBから取得
	⑤ 項目チェックし，表示項目を取得
	⑥ 全項目チェックするまで項目ごとに業務処理を繰り返し起動
	⑦ 出力帳票を編集し，印刷イメージデータ作成
	⑧ 表示情報・印刷イメージデータを端末で通信電文にて受信
	⑨ 端末に表示情報を表示
	⑩ プリンタから帳票出力 (イベント完了)

2. 対策手順

試験中のシステムは既に最終段階にあり、システム構造が変わるような対策手段は取れない状態にあったため、プログラム構造を変えずに性能改善の対策を行う手段を考える必要があった。

そこで、次のような手順で性能改善対策を行うこととした。

表4 性能改善対策手順

対策手順	対策手順項目内容
	① 一連の処理において、処理時間が遅い箇所の洗い出し
	② 処理時間が遅い箇所の関数レベルまでの絞込み
	③ 処理内容の分析
	④ 対策案の検討
	⑤ 現処理のベンチマークテスト実施
	⑥ テストプログラムへの対策案の反映
	⑦ テストプログラムによるベンチマークテスト実施
	⑧ テスト結果検討
	⑨ 開発環境システムへの対策の反映
	⑩ 開発環境システムでの性能試験
	⑪ 本番環境システムへの反映

評価基準は、今回の開発環境システムと本番環境システムにはマシン性能に差があり、本番環境システムが若干勝っていることから、開発環境システム上で性能が出せることを目標とした。

3. 調査・分析

洗い出しにおいてはいくつかのポイントに絞り洗い出しを行うこととした。
ポイントとして、以下の項目をあげた。

表5 調査・分析のポイント

	調査・分析のポイント
①	同一処理を繰り返している箇所はないか
②	不必要な処理（無駄な処理）を行っている箇所はないか
③	処理を短縮できる箇所はないか

3. 1 同一処理を繰り返している箇所の洗い出しの結果

(1) DBアクセスの回数

一括チェック処理ではすべての入力項目のチェック処理を実行しているが、入力項目チェックで関連情報をDBから取得する際に同一情報へのアクセスが繰り返し実行されていることがわかった。

単項目チェック処理においても、次項目の初期値表示のために関連情報を取得する際に同一情報へのアクセスが発生する場合がある。

3. 2 不必要な処理（無駄な処理）を行っている箇所の洗い出しの結果

(1) 初期化処理の重複

DBアクセスルーチンで関連情報取得エリアの初期化処理を重複して行っている箇所が発見された。

(2) ログの出力量

総合試験中だということもあり各処理においてログ出力を行っているが、出力量が膨大で全体処理時間に影響を及ぼしている。

3. 3 処理を短縮できる箇所の洗い出しの結果

(1) 広域領域の初期化

DBアクセスルーチンで関連情報取得エリアに広域の領域を確保している。この領域を調査した結果、1領域最大で30万Byteを使用しているケースが存在することが判明した。また、1万Byte以上の領域を使用しているケースを調査すると、更に多数の存在が確認できた。初期化処理にはシステム関数を使用しているが、byte単位で行っていることから検討が必要と判断する。

(2) 印刷イメージの帳票データ

帳票データ組立制御共通関数で印刷イメージデータを文字コードイメージで編集しているため、連続したスペースコードが数多く存在する。帳票の出力は、端末側に印刷イメージデータを通信電文として出力しているため、通信時間削減のためには印刷イメージデータを圧縮することで通信時間短縮を図れると判断できる。

(3) 関数内部変数定義

コンパイルオプションや変数定義の変更により処理時間の短縮が図れないか、現状の定義状況を調査する。

4. 対策検討

4.1 同一処理を繰り返している箇所の対策検討

(1) DBアクセスの回数

一括チェック処理においては、同一情報の取得を数十回、多いものでは百数十回アクセスしている。一度取得した情報をバッファ領域に保持し、同一の情報のアクセス要求にはバッファ領域内からデータを返すことで、DBアクセスに費やす処理時間を削減できることから、対策の必要があると判断する。

4.2 不必要な処理（無駄な処理）を行っている箇所の対策検討

(1) 初期化処理の重複

表6 初期化処理の重複例

処理の流れ	処理内容
↓	① 関数Aでエリアを確保し、初期化処理を行う
	② 関数Aが関数Bに確保したエリアのアドレスをパラメータで渡す
	③ 関数Bで受け取ったアドレスのエリアの初期化処理を行う
	④ 関数Bでエリアにデータを取得する

この例のように同一エリアに対して異なる関数でそれぞれ初期化処理を行っていた。初期化処理はどちらか一方でいいため、片方の初期化処理を削除することで無駄が省けるので対策の必要があると判断する。

また、文字列操作関数を使用しているケースでの初期化処理についても検討した。

表7 文字列操作関数の例

処理の流れ	処理内容
↓	① 関数Aで構造体エリアを確保し、初期化処理を行う
	② 関数Aが関数Bに構造体エリアのアドレスをパラメータで渡す
	③ 関数Bで文字列作成エリアを確保し、初期化処理を行う
	④ 関数Bで文字列作成エリアに文字列作成を行う
	⑤ 関数Bで構造体エリアメンバーに文字列作成エリアをコピーする

この例の場合では文字列の作成に `sprintf` 関数を使用し、文字列作成エリアのコピーに `strcpy` 関数を使用している場合は、関数Bにおける初期化処理は行わなくても処理結果に問題は起こらない。このことから関数Bの初期化処理は削除できると判断する。（文字列編集の `sprintf` 関数は文字列の末尾に `Null` 文字

を付加し、文字列操作の `strcpy` 関数は `Null` 文字までの文字列をコピーできるので問題ないが、メモリ操作の `memcpy` 関数は指定バイト数のコピーで文字列の末尾を判定できないため、文字列のコピーに `memcpy` 関数を使用している場合は削除不可能)

(2) ログの出力量

出力しているログの内容を検討した結果、次の種類のログ内容が確認できた。

表8 ログ内容

	確認できたログ内容
①	関数実行を確認するために入口、出口で出力している場合
②	関数の入力パラメータの内容を出力している場合
③	判定文 (<code>if</code> 文) で判定しているデータ内容を出力している場合
④	異常発生時の異常内容を出力している場合
⑤	計算経過を随時出力している場合
⑥	処理の流れ (ルート) を確認するために出力している場合

総合試験中だということもあり各処理においてログ出力を行っているが、出力関数の呼出回数・出力処理時間を測定した結果、量が膨大で全体処理時間に影響を及ぼしていることが判明したため、障害発生時の原因究明に必要な内容を除いてはログ出力を極力抑えることとした。

4. 3 処理を短縮できる箇所の対策検討

(1) 広域領域の初期化

領域の初期化処理を調査した結果、1万Byteから最大30万Byteの広域領域に対してシステム関数を使用して初期化処理を行っていることが判明した。

使用しているシステム関数を検討した結果、システム関数はbyte単位で処理していることから、32bit単位で処理することで短縮できないかと考え、テストルーチンを作成し、30万Byteの領域初期化の処理時間を測定したところ、約25%の削減が可能であることが確認できた。この結果から32bit単位の初期化処理に置き換えることは有効であると判断する。

(2) 印刷イメージの帳票データ

帳票データ組立制御共通関数で作成した印刷イメージデータから連続したスペースコードをエスケープシーケンスコードの水平アドレスタブに変換する変換関数をテストプログラムで作成した。水平アドレスタブの変換には5byte必要となるため、6byte以上のスペースコードが連続している場合を対象として変換する。

テストプログラムで変換した結果、平均で約45%が削減できた。この結果から印刷イメージデータを圧縮することで通信時間短縮を図れると判断する。

(3) 関数内部変数定義

コンパイルオプションの変更は、すべてのプログラムに対してコンパイルし直さなければならないこととマシンコードレベルにおいての検証が困難なことから工程的に

は合わないため、コンパイルオプションの変更は行わないこととする。各関数内で定義している変数の宣言に `register` 宣言をしていないことが調査の結果わかった。変数の宣言に `register` 宣言をすることにより実行レベルで処理時間の短縮が図れることから対策の対象とする。

5. 改善対策

対策ポイントとなる項目を検討した結果、次の項目に対して対策を行うことになった。

5. 1 同一処理を繰り返している箇所の改善対策

(1) DBアクセスの回数削減

ファイルアクセスルーチンの処理において、アクセス回数を削減するためにバッファ領域を用意し、アクセスして取得したデータがバッファ領域に存在するかを検索して、無ければバッファ領域に登録し、在ればファイルアクセスを行わずにバッファ領域内からデータを返すように修正する。それぞれの情報に対して異なるキーでアクセスする場合を検討して、バッファ領域を設定する。

今回は、各情報に対して最大5情報分の領域を確保した。また、すべての情報に対して領域を確保するのではなく、取得する関連情報のアクセス回数を調査した結果、数十回以上アクセスしている情報の回数の多いものから6つを対象として対策した。

表9 DBアクセス

処理の流れ	処理内容
↓	① バッファ領域を検索し、無ければDBから取得
	② 取得結果をバッファ領域に登録
	③ バッファ領域を検索し、在ればバッファ領域から取得

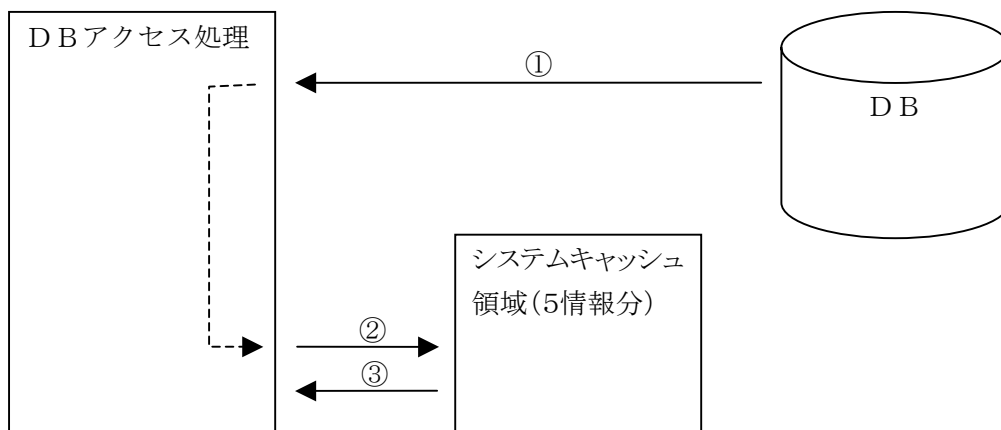


図3 DBアクセス

5. 2 不必要な処理（無駄な処理）を行っている箇所の改善対策

(1) 初期化処理の削除

連続した処理の流れの中で異なる関数で同一のエリアに対して初期化処理を行っている場合の片方のエリアの初期化処理を削除した。5. 2. (1) の場合は関数Bの初期化処理を削除した。また、文字列関数を使用する場合で初期化処理を削除して問題ない場合は削除した。

(2) ログ出力の削除

総合試験中のため、処理確認に必要なログ出力ではあるが、出力内容によっては必ずしも必要とならない内容も多数あるため、障害発生時の原因究明に必要な不可欠な内容を除いて極力削除した。

5. 3 処理を短縮できる箇所の改善対策

(1) 高速初期化処理への切り換え

テストプログラムによる検証の結果、有効性が認められたことから、1万byte以上の広域のメモリ領域の初期化処理をシステム関数から32bit単位の初期化処理に置き換える。

(2) 出力帳票イメージデータの圧縮

テストプログラムによる検証の結果、有効性が認められたことから、帳票データ組立制御共通関数で編集した印刷イメージデータをエスケープシーケンスコードに置き換え編集する関数で編集したデータを通信電文として送信するように修正した。

(3) 関数内部変数定義変更

各関数内での変数定義の中で、char, short, intなどの変数の定義をregister宣言するように修正する。すべての関数の宣言を修正することは困難であるため、今回の性能改善対策を行う関数と特に使用頻度の高い共通関数についてのみ対策するものとする。

6. 結果（効果）

対策の結果、以下のような処理時間となった。

表10 対策結果

	単項目チェック	一括チェック
対策前	約2～5秒	約10～20秒
目標値	1秒以内	5秒以内
対策後の開発環境システム	約1.1秒（平均値）	約5.1秒（平均値）
対策後の本番環境システム	1秒以内	5秒以内

今回の対策は最終調整段階の作業フェーズであり、システム稼動までにすべての処理に対して対策を施すことは工程上にも余裕が無かったため、各業務処理から呼び出される共通処理部に対してのみ実施している。

対策はオンライン業務処理を対象として実施した結果であるが、バッチ業務処理においても対策した共通処理部を使用しているため、バッチ業務処理の全体処理時間の約50%が削減された。

7. むすび

7. 1 目標達成について

対策の結果は開発環境システムで、ほぼ目標値に近い結果を出すことができたことから本番環境システムにおいて目標値をクリアする結果が得られた。

7. 2 改善対策の有効性について

今回の改善対策内容は、システムのすべての処理に対して対策を施すことができなかつたにも係わらず、性能向上に繋がったことから有効な手段であったといえる。また、システムの処理構造を根本から変えることができないことを考慮して検討した対策内容になっているため、今回のシステムに限らず、他のシステムへの流用も十分可能だと考える。

7. 3 今後の展望について

今回実施した性能改善対策は、特別なことを行っているわけではなく、ごく一般的な手法であると思うが、近年のハードウェアの処理性能の向上は著しいものがあり、ハードウェア性能に任せた処理作成になっているケースが少なからず存在している。また、ソフトウェア開発の工程短縮が叫ばれている中で既存システムからの流用、変更で開発工程の短縮を行うこともある。今回の結果は性能改善目標値に対しては達成したといえるが、すべての業務処理に対策を施すことができれば、更に高い性能結果が得られたはずである。

今回のシステムに限らず、どのようなシステム開発においても同じことがいえると思うが、今後は設計フェーズの段階で十分な検討をし、後戻り作業が発生しないように心がけたい。

最後にここで紹介した性能改善対策内容は自社内の作業ではないため、詳細な数値データは出さずに概略的な数値での結果に留める。