オブジェクト指向技術による
Javaフレームワークの構築

中村伸裕（なかむらのぶひろ）

〔略　歴〕

1988年　住友電気工業株式会社入社

現在　　情報システム部　システム技術課　主査

　　システム経験年数　11年

谷本収（たにもとおさむ）

〔略　歴〕

1986年　住友電気工業株式会社入社

現在　　住友電工情報システム株式会社

　　システム開発部　商品開発課　課長

　　システム経験年数　13年

〈要　　約〉

当社ではデータ中心設計やＲＡＤ開発手法の導入、プログラムの部品化によってシステム開発のコスト削減、納期短縮、品質向上に取り組み、効果を上げてきた。しかし、経営を取り巻く環境が厳しくなり競争激化によるビジネスサイクルの短期化やコスト競争などによりシステム構築のより一層の納期短縮やコスト削減が求められるようになってきた。

そこで我々はオブジェクト指向技術を導入し、プログラムの再利用によって各開発プロジェクトで作成するプログラムを最小限にし、納期短縮、コスト削減、品質向上を図ることにした。従来のプログラム言語では部品化の対象はアプリケーションから独立して切り出せる部分であり、業務ロジックを含むようなプログラムは部品にすることができなかった。そのため、システムの部品化は全体の２割程度にとどまっていた。オブジェクト指向言語ではメインルーチンや処理ロジックを含んだ再利用可能なプログラムすなわちフレームワークを構築しておくことで、差分プログラミングによりシステムを開発することが可能となり、部品化の範囲を拡大することができる。

フレームワーク構築のためにはできるだけプログラムを抽象化し、部品化を進める必要がある。しかし、多くのプログラムにはそのプログラムで使用する項目名称等の固有名詞が埋め込まれており、作成したプログラムを再利用することは難しい。そこで、データ項目特有の項目名称、データ長、データ型、エラーチェック方法等をカプセル化した項目オブジェクトを考案した。この項目オブジェクトを使用することで項目特有の処理（入力桁数の設定や、数値チェック等）をプログラムから分離し、部品化の範囲を大幅に広げ、フレームワークの構築を可能にした。また、数百から数千に及ぶ項目オブジェクトはＤＢの設計情報から自動生成できるツールを作成し、項目オブジェクトの実装を可能にした。

フレームワークを検証するため、パイロットシステムで試行してみることにした。その結果、従来の開発言語に比べ50%の工数削減、50%の納期短縮が実現でき、フレームワークの有効性が確かめられた。また、パイロットシステムの開発により、再利用される部品の品質管理が重要であることがわかり、管理ツールを作成して単体テストの徹底を図った。また、ソースのバージョン管理を行うことで万一不具合が発生しても修正モジュールが素早くリリースできるようにした。

以上の取り組みによりベースとなるフレームワークを構築することができた。現在、このフレームワークを使って購買管理、物流管理、生産管理等のシステム開発を行っている。今後は、フレームワークを更に充実させ生産性の向上を図っていきたいと考えている。

目　　次

1. はじめに …… 4
2. Javaフレームワーク構築の背景と狙い …… 4
   2.1 Javaフレームワーク構築の背景 …… 4
   2.2 Javaフレームワーク構築の狙い …… 4
3. Javaフレームワークの検討 …… 5
   3.1 イントラネットでの生産性悪化の要因 …… 5
   3.2 フレームワークに必要な機能の検討 …… 6
   3.3 プログラム再利用の形態 …… 7
4. Java部品の開発 …… 7
   4.1 必要な部品の抽出 …… 7
   4.2 項目オブジェクト …… 9
   4.3 ツールによる項目オブジェクトの生成 …… 10
   4.4 Javaの応答時間の検証 …… 10
   4.5 部品の開 …… 10
5. フレームワークの構築の取り組み …… 12
   5.1 基本フレームワークの構築 …… 12
   5.2 再利用可能なプログラムの作成 …… 12
   5.3 フレームワークの基本サブシステム …… 14
6. フレームワークの適用 …… 15
   6.1 プログラマーの養成 …… 15
   6.2 フレームワークの品質管理 …… 15
7. 評価と今後の課題 …… 16
   7.1 評価 …… 16
   7.2 今後の課題 …… 17
8. おわりに …… 17

１．はじめに

当社ではデータ中心設計やＲＡＤ開発手法の導入、プログラムの部品化によってシステム開発のコスト削減、納期短縮、品質向上に取り組み、効果を上げてきた。しかし、経営を取り巻く環境が厳しくなり競争激化によるビジネスサイクルの短期化やコスト競争などによりシステム構築のより一層の納期短縮やコスト削減が求められるようになってきた。

そこで我々はオブジェクト指向技術を導入し、プログラムの再利用によって各開発プロジェクトで作成するプログラムを最小限にし、納期短縮、コスト削減、品質向上を図ることにした。本論文ではその取り組みについて紹介する。

2．Javaフレームワーク構築の背景と狙い

2.1　Java フレームワーク構築の背景

当社では1995年よりデータ中心設計手法を導入し、外部設計から総合テストのフェーズで30%の生産性向上を達成した。また、一部のシステムではRAD(Rapid Application Development)開発手法を導入することでプロトタイピングを活用し、主に外部設計、詳細設計の期間短縮、コスト削減に成功し、プログラム作成のフェーズがシステム開発全体の中で大きなウエイトを占めるようになっている。

また、1997年以降、基幹システムの開発プラットフォームにイントラネットを採用し、米国Allaire社のアプリケーション・サーバー Cold Fusion を使ってシステム構築を行ってきた。Cold Fusion はHTMLを拡張したタグ言語であり、言語の習得が容易であり、イントラネットの開発ツールとしては高い生産性を上げていた。しかし、大規模システムの開発においては部品化の機能が弱くこれ以上の生産性の向上が望めない状況になっていた。

一方、応答速度が遅いため実用化が進まなかったJava言語はJava VM (Virtual Machine)をサーバー側で動かすServletの形態で米国を中心に実用化が進んでいた。Java言語は以下の特徴を持っており、将来性が期待できるため採用することにした。

(1)　幅広いプラットフォームで動作し、次世代OSとして期待されるLinuxへの対応も容易である。
(2)　多くのベンダーが共通のAPI仕様でツールを提供しており、１社のツールベンダーに依存することない。
(3)　オブジェクト指向言語であり差分プログラミング等のプログラム再利用の仕組みを持っている。

2.2　Javaフレームワーク構築の狙い

Java言語はオブジェクト指向言語であり、部品とフレームワークの２つの方法でプログラムの再利用が可能である。部品化は従来から行われている手法であり、複数のプログラム中で共通部分を抜き出し、他のプログラムから呼び出して再利用するものである。一般的に部品は業務ロジックから独立したものであることが多い。

一方、フレームワークはメインルーチンや業務ロジックを含むプログラムの集まりである。例えば受注フレームワークは、一般的な受注処理を実装しており、そのまま利用するか差分プログラミングによって独自の機能を追加して利用することができる。フレームワークを構築することにより従来の部品化の範囲を超えて広い範囲でプログラムの再利用が可能になる。

今回の取り組みでは、これらの特徴を活かし以下の目標を掲げてフレームワークの構築に取り組むことにした。

(1) プログラム開発の生産性30% 向上
部品を整備することでアプリケーション部分のコード量を50%削減し、生産性の向上を図る。

(2) プログラム開発の納期30% 短縮
複数のシステムで必要なユーザー登録、権限管理、印刷管理等のサブシステムをフレームワーク化することでシステムごとの開発を不要にし、納期短縮を図る。

(3) プログラム品質の向上
システムごとに記述するコード量を削減し、バグの絶対量を減らす。

3．Javaフレームワーク構築の取り組み

3.1 イントラネットでの生産性悪化の要因

当社で開発するJavaフレームワークはイントラネットでの使用を前提としているため、イントラネットの形態がプログラム開発の生産性にどのような影響を与えるか調べることにした。図１にイントラネットのシステム構成を示す。ブラウザーとアプリケーション・サーバーの間はセッションレスのHTTPプロトコルが使用されており、ブラウザーに表示される画面はHTMLで構成される。HTTP及びHTMLは情報を閲覧するために開発された仕様であり、基幹系システムを構築する場合は、生産性を悪化させる要因となることがわかった。

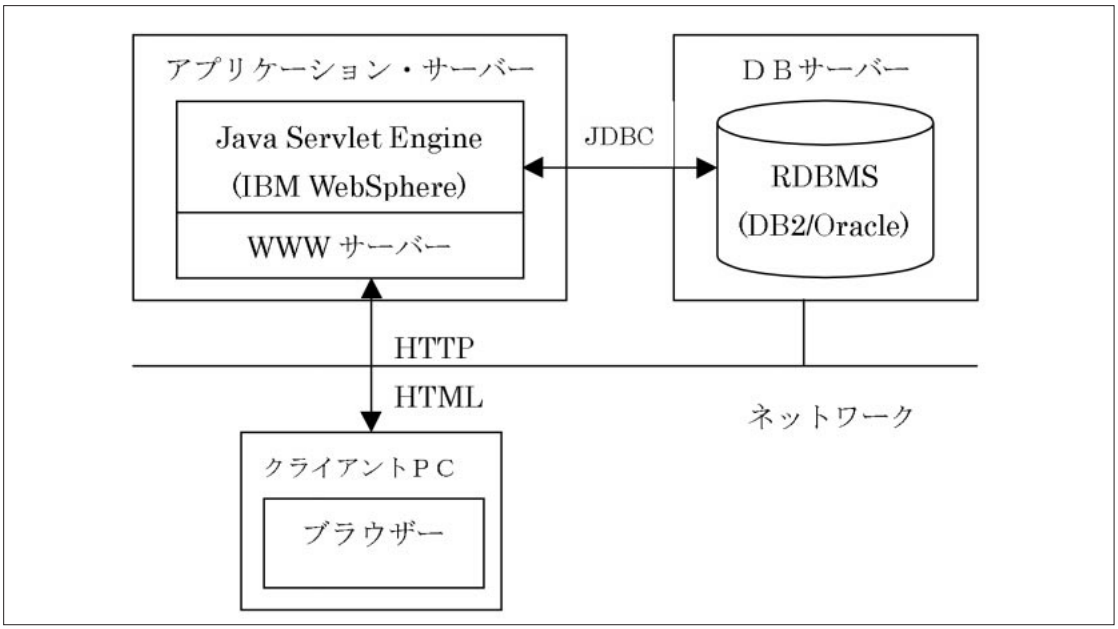

図１．イントラネットのシステム構成

(1)　HTTPプロトコルによる生産性の悪化

イントラネットで使用するHTTPプロトコルがセッションレスであるため、ブラウザーとサーバー間の通信は毎回切断されている。しかし、基幹システムではログインしたユーザー情報をページ間で引き継ぐ必要があり、セッション管理の仕組みを作り込む必要がある。また、画面遷移の順番が保証されないため、直接URLを指定して途中の画面やアクセス権限のない画面に接続することが可能になる。各プログラムでは画面遷移が設計した順番でアクセスされているかのチェックや権限チェックを行う必要がある。その他、登録ボタン(SUBMIT)をダブルクリックするとデータが二重登録されることがある。データがサーバーに二重送信されるためであるが、これを防ぐ仕組み等も必要となる。

(2)　HTMLによる生産性の悪化

クライアント＆サーバーシステムではＧＵＩによって画面設計を行うが、イントラネットでは手作業でHTMLを出力するコーディングを行わなければならない。すなわち、画面で使用するデータ項目すべてに対して項目名称や入力形式（テキストボックス、ラジオボタン等)、入力桁数の指定等をソースコード中で行わなければならない。また、HTML出力のコードと業務ロジックのコードは混在することが多く、部品化を妨げる要因になっている。

これらの要因を考慮しながらプログラムを作るのは生産性が大きく低下するだけでなく、考慮漏れによる品質低下の恐れがある。そこで、フレームワークの内部でこれらの問題を解決し、プログラマーが特に意識しなくても高品質のプログラムが作れるようフレームワークの構築を行うことにした。

3.2　フレームワークに必要な機能の検討

構築するフレームワークは今後当社で開発する全てのシステムのベースとなるものである。リリース後の大きな変更はできるだけ避けたい。フレームワークが今後開発するいかなるシステムにも対応できるようフレームワークとして持つべき機能を検討した結果、下記の機能を実装することにした。

(1)　イントラネット固有の問題の隠蔽

3.1章で述べたイントラネットでの生産性悪化の要因を隠蔽し、プログラマーが特に意識することなくプログラムの開発ができるようにする。

(2)　多国語対応

グローバル化に伴い海外関係会社と国内で同一システムを利用するケースが増えつつある。日本語版のプログラムを英語に翻訳してプログラムを二重に持つのは保守性を悪化させるため、パラメーター指定で日本語、英語の表示を切り替える仕組みを実装することにした。この機能は人件費の安い海外でシステムを開発し、日本語化する際にも利用する予定である。

(3　項目名称のカスタマイズ

当社ではパッケージソフトの外販を行っているが、項目名称のカスタマイズのニーズが強く、フレームワークの基本機能として実装することにした。

3.3　プログラム再利用の形態

プログラムの再利用を考えた場合、プログラム全体をそのまま利用するものや、データ加工のための単純なサブルーチンまでさまざまな粒度のものがある。当社では再利用の形態を以下の４階層に分類することにした。

(1)　部品：プログラム再利用の最小単位であり、業務アプリケーションから呼び出して使用する。

(2)　プログラムパターン：部品を組み合わせた業務ロジックを含まないプログラムであり、使用するデータ項目等をパラメーターで与えることができる。基本テーブルのメンテナンス等に使用可能である。

(3)　フレームワーク：部品やプログラムパターンを組み合わせ、基本的な業務ロジックを組み込んだ再利用可能なプログラムの集合体である。プログラムは使用するシステムに合わせて差分プログラムによって再利用される。

(4)　パッケージ：システム全体をそのまま再利用する形態である。

今回の取り組みではまず部品を開発し、プログラムパターン、フレームワークの順に開発を進めることにした。

4．Java部品の開発

4.1　必要な部品の抽出

フレームワークの構築に必要な部品の抽出するために、Servletの内部処理と出力されるHTMLページの解析を行った。

(1)　Servlet 内部処理の解析

業務システムのServlet内部処理を解析した結果、非常に単純な構造になることがわかった。その構造を図２に示す。ブラウザーからの要求でサーバー側のJavaプログラムが起動する。このプログラムは擬似的に作成したセッション変数からログイン情報や前の画面の情報を取得し、アクセス権限や画面遷移の正当性を検査する。次にブラウザーから送信されるデータ（例えば入力された受注番号等）を取得し、その値を元にＳＱＬを組み立て、ＳＱＬを実行する。検索画面であれば検索結果をフォーマット変換等のデータ加工処理を行った後、HTMLタグと共にブラウザーに送信する。最後に次回のアクセスのためにセッション変数を保管しておく。CSVファイルを出力しExcelと連携するようなプログラムを除けば大半が上記の順序で処理を行うことになる。図２の各処理単位を基本部品として必要な数の部品を用意すればプログラマーは該当する種類の部品の中から必要な部品を見つけだして組み合わせることができる。また、全体をフレームワークとして用意することで、必要な業務ロジックを追加するだけでプログラムを完成させることができる。

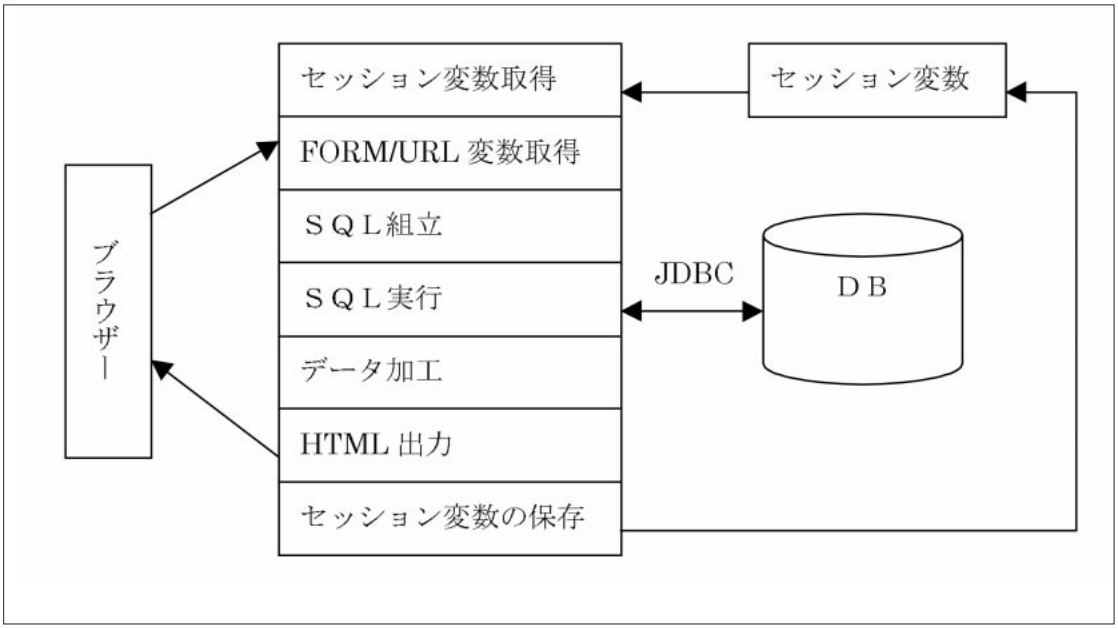

図２．Servletの内部処理

(2) HTMLページの解析

図３にイントラネットシステムのサンプル画面を示す。

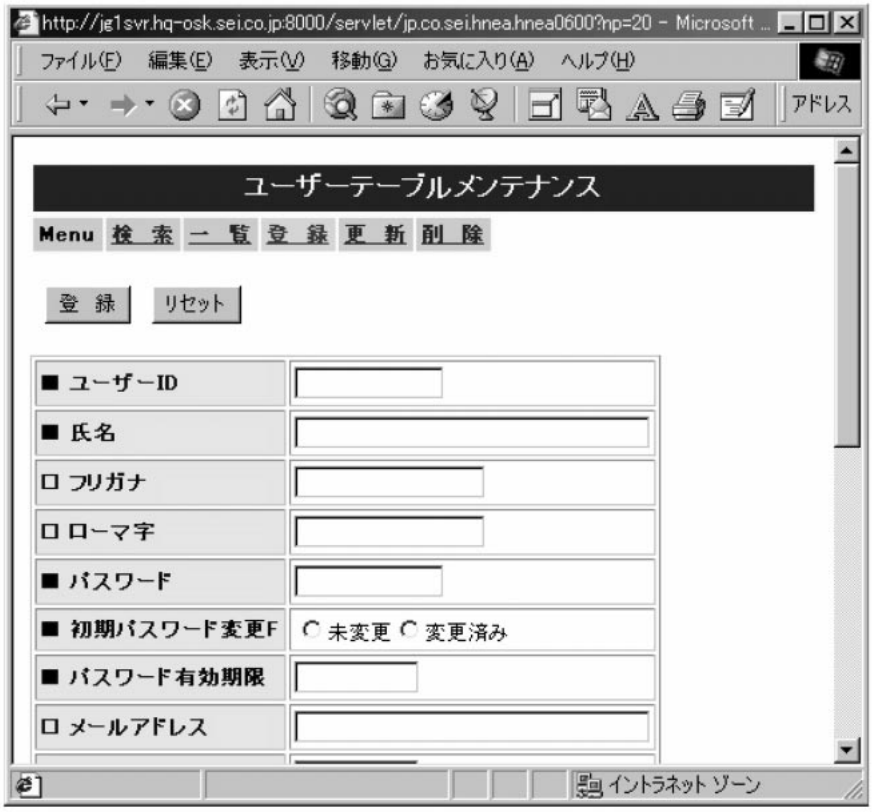

図３．イントラネットシステムのサンプル画面

この画面は、タイトル、インラインメニュー、入力フォーム等のHTML部品に分解することができる。しかし、この画面で主となる入力フォームの部分を部署登録の画面で再利用することは難しい。ユーザー登録のプログラムにはユーザーＩＤ、氏名といったプログラム特有の項目名称が埋め込まれているからである。部品化を推進するためにはプログラムからデータ項目の属性を分離する必要があると考え、実装方法について検討することにした。

4.2　項目オブジェクト

プログラムからデータ項目を分離するために項目オブジェクトを考案した。項目オブジェクトは表１に示す属性を持つオブジェクトである。

表１．項目オブジェクトに実装する主な属性

項目名	コーディングで使用するデータ名称。通常、ＤＢの列名を使用。
項目名称	日本語及び英語の項目名称。画面表示に使用。
対応DB列名	対応するＤＢの列名
データ型	文字型、数値型、日付型等の区別
入力データ長	データの最大長
入力形式	テキストボックス、選択ボックス、ラジオボタン等の区分
入力可能な値	選択入力を行う際に、一覧に表示する値
ヘルプメッセージ	入力の際に表示する項目の説明
エラーチェック	エラーチェックのロジックの実装
必須区分	入力項目が必須入力であるかの区分
その他	項目毎の独自処理を行うフィルタープログラム等

項目オブジェクトを使用した画面出力の動作を図４に示す。メインプログラムには画面で表示する項目が設定されている。ブラウザーに入力画面を表示するFORM部品はパラメーター userid, username,... に対応する項目オブジェクトを探し出す。データ名称やデータ長等は項目オブジェクト

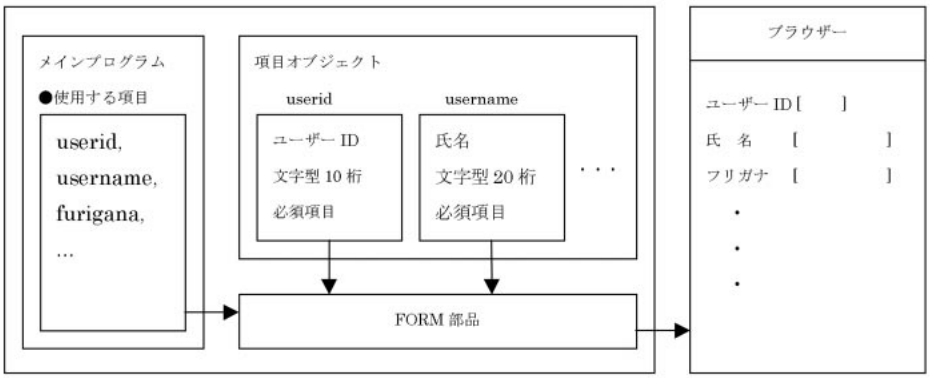

図４．項目オブジェクトの動作

に保持されており、FORM部品は項目オブジェクトからこれらの情報を取得し、ブラウザーに入力

画面を表示する。このメインプログラムにはデータ名称やデータ長等の情報は一切コーディングされていない。従って、このプログラムのパラメーター定義部分を {"userid", "username", "furigana", ...} から、{"deptno", "deptname", ...} に変更するだけで部署登録の入力画面に変更することが可能となる。
このように項目オブジェクトによりプログラムからユーザーインタフェースを分離し、部品化することに成功した。さらに項目オブジェクトにより項目のエラーチェックやDBの更新等も部品化することが可能になった。
項目オブジェクトを実装することで応答速度が低下する懸念があったので、一度使用した項目オブジェクトはメモリー中に常駐し、再利用できるようにした。その結果、項目オブジェクト利用による応答時間の悪化は10ms以下となり問題なく使用できることを確認した。

4.3　ツールによる項目オブジェクトの生成

項目オブジェクトによりHTML部品が実現できることがわかった。しかし、通常のシステムでは数百から数千のデータ項目があり、手作業で作るのは困難である。そこでＤＢの定義情報から項目オブジェクトを生成するツールを作成した。このツールはＤＢに定義されているデータ型とデータ長を取り出し、入力値に対する数値チェック、日付チェック、データ長のチェック等を行うJavaプログラムを生成する。入力形式は使用頻度の高いテキストボックスをデフォルトで設定し、ラジオボタン等の他の入力形式を使う項目は後で簡単に変更できるようにした。このようにして、大量の項目オブジェクトが短時間で実装できるようになった。

4.4　Javaの応答時間の検証

部品の開発に当たってはJavaの動作速度を確認した。速度が遅ければ部品は動作速度の速い単機能の部品を作る必要がある。速ければ多機能部品を作ることが可能である。動作速度を検証するために小規模なWindows NT Server (CPU: Pentium 266MHz) 上のJava Servlet と RDBMS (DB2 UDB) を使用した簡単な検索プログラムを作成したところ平均800ms で応答が返ってくることがわかった。この応答時間は複数ユーザーが同時に使用することを考えた場合じゅうぶんな速度とは言えない。そこで800msの内訳を調べたところ、DBコネクションの確立に約700ms、検索に約50ms、その他内部処理に50msであった。そこでプログラム（Servlet）単位でコネクションを確立するのではなく、予め確立したコネクションをプールしておき、必要に応じてプログラムからコネクションを取得する方式に変更することで、平均約150msの応答時間が確保できるようになった。オンラインの最低応答時間を３秒に設定すると、同時に20名がアクセスしても耐えられる速度でありJavaが高速に動作することを確認した。

4.5　部品の開発

部品の開発においてはプログラマーが簡単に部品を利用できることを一番に考えた。Java言語の習得はスクリプト言語に比べ難しく、Java プログラマーを短期間で養成するためには部品理解の負担をできるだけ減らす必要がある。また、プログラマーが部品の存在を知らずに新たに部品を作成す

れば生産性の向上は期待できない。そこで、以下の方針で部品の開発を進めることにした。

(1)　個々の部品を多機能にし、作成する部品の数を極力減らす。
(2)　作成すべき機能に対して使用する部品が１つ決まるようにし、プログラマーが誤った部品を使わないようにする。（似たような部品は作成しない）
(3)　プログラマーが多くの部品を共通の呼び出し方法で利用できるようにする。

今回の取り組みでは70個の部品を作成した。部品の中で定義されたメソッド（関数）は約1,000であり、プログラムソースは約21Kステップであった。この中にはプログラマーが意識する必要のないセッション管理等の部品やクラス設計の過程で生じた内部部品が含まれており、プログラマーが主に使用する部品は表2に示す33個の部品である。この内13は基本部品を組み合わせたプログラム・パターン(以下、パターン)であり、一般的な画面で使用する部品はDB部品、データ加工部品、HTML部品の12個に限定した。
検索画面を作成する場合、プログラマーはもっとも簡単にプログラムを作成できるパターンが利用できないか検討する。パターンを使用すれば画面表示とDB操作が部品の中で行われるため、プログラマーのコーディング量は大幅に削減される。パターンが使用できない場合は、データ１件を明細表示するか、複数件を一覧表示するかによって２つある部品のうち、いずれかを選択することになる。次に部品のプロパティ（表示項目、リンクの設定、表示色等）をセットし、目的のプログラムを完成させる。
このようにプログラマーが使用する部品の数をできるだけ少なくし、幅広い用途に対応できるよう豊富な機能を持たせることでプログラマーが容易に必要な部品を選択できるようにした。

表２．プログラマーが主に使用する部品

種　類	個数	内　容
FORM部品	2	FORM／URL変数の取り込み、エラーチェック等
DB部品	3	DB検索、DB更新、
データ加工	2	データのハンドリング、フォーマット変換等
HTML部品	5	検索結果の一覧表示、明細表示、 明細入力画面、一括入力画面、 汎用HTML出力
Excel連携	4	CSVファイルのダウンロード、アップロード、DBへの登録
パターン	１３	検索、登録、変更、削除、一括登録、サブウインドウ等のロジックを含む機能部品
その他	4	メール送信、デバッグ用ツール、部品作成用の部品他
合計	３３	

5．基本フレームワーク

5.1　フレームワークの構築

図５は基本フレームワークの構造を示したものである。基本フレームワークではセッション変数やＤＢコネクションの管理、権限管理等の処理を実装しているが、メイン処理は何も実装されていない。開発するプログラムはメイン処理部分を差分プログラミングにより実装するため、セッション管理やコネクションの管理を特に意識する必要はない。また、FORM変数の取得やＳＱＬ実行、HTML出力に関してはいくつかの部品が用意されており、その中から必要な部品を選択して使用することでメイン処理のコーディング量を大幅に削減することができる。

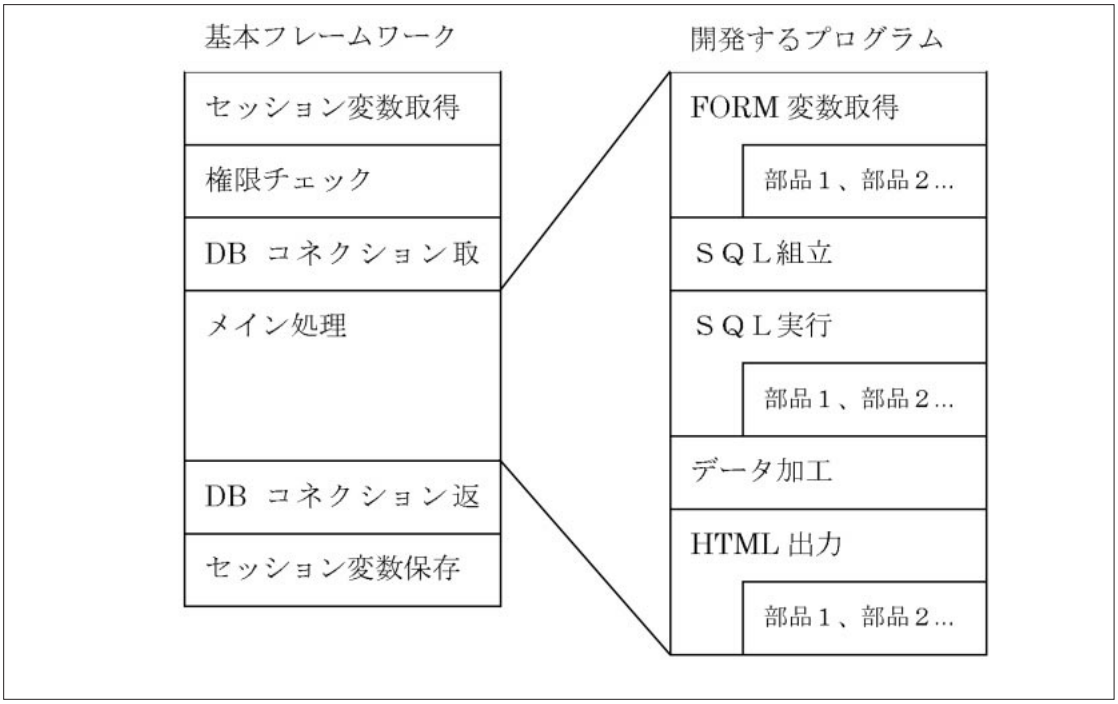

図５．基本フレームワークの構造

5.2　再利用可能なプログラムの作成

フレームワーク構築のポイントはプログラムの再利用であるが、オブジェクト指向言語の特徴である継承による差分プログラミングには欠点がある。継承では元となるプログラムの一部を上書きして新しいプログラムを追加するが、その為には継承元のプログラムの内部構造をすべて理解しなければならない。また、新しく作成したプログラムは継承元のプログラムが修正された場合、正常に動作する保証はない。例えば上書きしたメソッド（関数）の名前が継承元のプログラムで変更された場合、作成したプログラムは動作しなくなる。従って、継承による差分プログラミングは保守性を考慮して適用していく必要があり、業務改善に合わせてメンテナンスしていく必要のある業務プログラムへの適用は難しいと考えられる。また、プログラム本数が数百本～千本のシステムではＳＥ、プログラマーの人数も多く、継承による差分プログラミングを考慮しながらプログラムの設計・製造を行うことは優秀な人材の確保および開発納期の面でも難しい。そこで我々はプログラム

をモジュール化し、必要なモジュールを自由に組み合わせてプログラムを開発・変更できる仕組みを開発した。

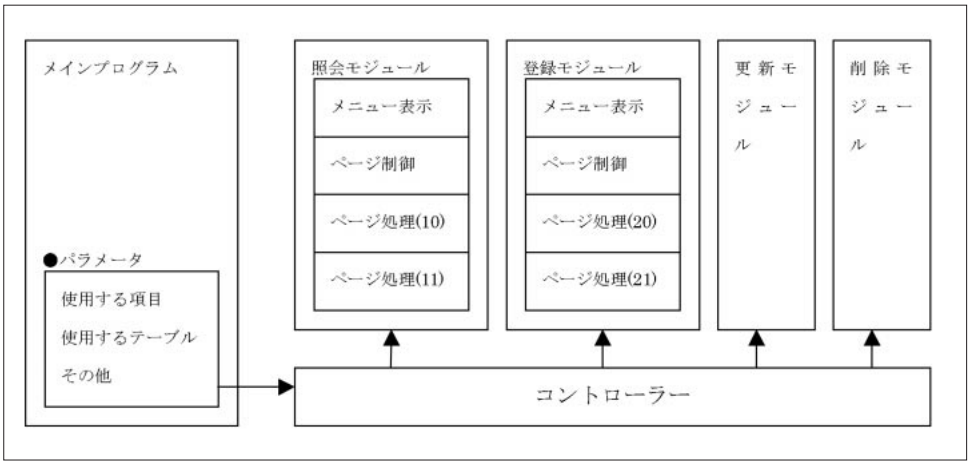

図６．プログラム再利用の仕組み

図６は図３に示す画面の内部構造を示したものである。メインプログラムではプログラムで使用するデータ項目や対象となるテーブルがパラメーターとして設定されている。処理を行うプログラムは機能毎に照会、登録、変更、削除に分割されている。図６では紙面の都合で更新、削除の図が省略されているが照会、登録と同じ形式のモジュールである。照会プログラムではメニュー表示、ページ制御、ページ処理の処理が実装されている。図３に示す画面上部のメニューで「検索」「一覧」部分がこのモジュールのメニュー表示に該当し、ブラウザーには全モジュールのメニューが合成されて表示されている。ブラウザーに表示される画面は全てページ番号がつけられており、このモジュールでは検索キーの入力画面がページ番号10であり、入力されたキーで検索した結果を表示する画面がページ番号11である。複数のモジュールはコントローラーによりブラウザーから送られてくるページ番号で制御される。このようなモジュール構成でプログラムを開発すると以下のメリットが得られる。

(1)　モジュールが処理単位で独立しているため作成が容易であり、メンテナンスも容易になる。
(2)　必要な機能のモジュールを自由に組み合わせて使うことができる。
(3)　モジュールの入れ替え・追加によりカスタマイズが可能である。

例えば、検索キーの入力画面(ページ番号10)はそのまま使用し、検索結果にグループ集計を表示する新しいプログラムを作成する場合、ページ番号11の処理を実装したモジュールを差し込むだけで新しいプログラムを作成することができる。図７は新しいモジュールを差し込んだものである。コントローラーは最初に登録されたページ処理を呼び出す為、ベースとなる新照会モジュールのページ処理(11)が実行される。

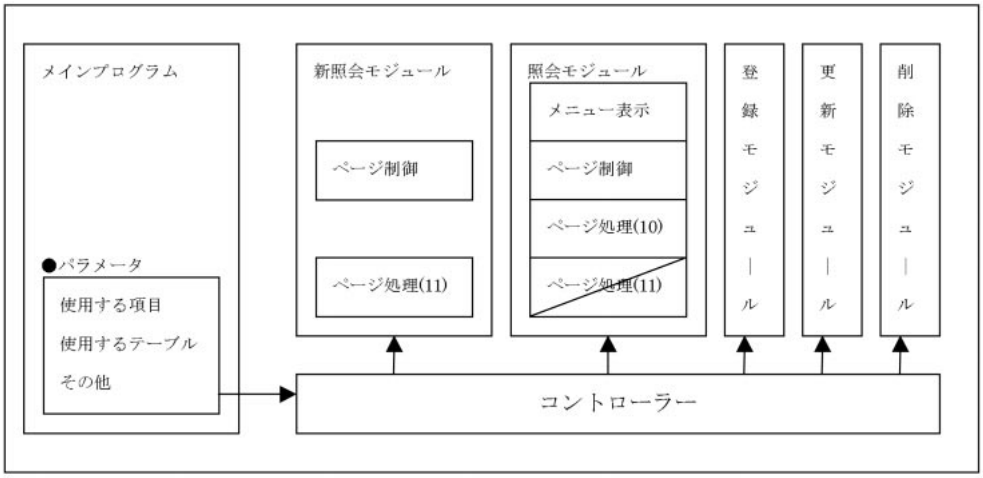

図７．カスタマイズされたプログラムの構造

この構成では、照会モジュールのページ処理(11)が変更されても新照会モジュールは影響を受けない。このように独立性・保守性の高い業務プログラムが差分プログラムによって効率よくカスタマイズできる仕組みを築いた。

5.3　フレームワークの開発

受注・出荷、購買管理等のフレームワークは個別プロジェクトとして進めることにし、本取り組みでは、社内の開発プロジェクトで共通的に使用でき、使用頻度の高い下記のフレームワークを開発することにした。これらのフレームワークにより、各開発プロジェクトでのこれらの機能を個別に開発する必要がなくなり、開発工数の削減が可能となる。

(1)　ユーザー管理

システムを利用するユーザーや部署、役職等の管理機能を提供する。当社では分社化が進んでおりシステムの利用者に関しては子会社社員の比率が高まっている。その為、従来のように人事ＤＢからユーザー登録することができない。システム管理者の負担を減らすため、利用者本人にユーザー登録申請を行わせシステム管理者が承認する仕組みをつくり、システム管理者がユーザー情報を入力しなくてよいようにした。また、今後業務用アプリケーション・サーバーが増加することが見込まれているため、１回のログインで複数の業務サーバーを利用できるシングルサインオンの機能も提供する。

(2)　メニュー・権限管理

ユーザーもしくはグループ毎に使用可能なアプリケーションを登録しておき、メニューの制御を行う。また、URLを直接指定して権限のない画面にアクセスすることを防止するための機能を提供する。

(3)　帳票管理

システムから出力する帳票を一時保管し、プリンターで用紙が詰まった場合の再印刷機能やブラウザーでの閲覧機能を提供する。

(4)　ワークフロー

最近ではワークフローによる業務の効率化を実現するシステムが多い。このフレームワークはワークフローエンジンであり、ワークフローのシステムを容易に構築するための機能を提供する。

6．フレームワークの適用

6.1　プログラマーの養成

Java言語を本格的に導入するためには、短期間でシステム開発に必要な人数のプログラマーを養成する必要がある。しかし、一般的にオブジェクト思考言語の習得は難しいと言われており、特に部品やフレームワークが設計・開発できる上級プログラマーの養成は非常に難しい。そこで、本社に基本フレームワーク及び基本部品の開発チームを設置し、社内のシステム開発プロジェクトにそれらを配付することにした。各プロジェクトは開発チームに要求を出し、開発チームは全体のバランスを取りながら基本フレームワーク及び基本部品を拡張していく体制にした。このような開発体制により、各プロジェクトのプログラマーは主に基本フレームワーク及び基本部品の使い方を習得するだけでプログラムが作成できるようになった。Javaネイティブの難しいコーディングは基本部品で吸収したため、膨大なJava のAPI をすべて覚える必要もなくなった。その結果、Java言語の基本講習0.5日、基本部品、基本フレームワークの演習を2.5日、合計３日間の講習でプログラマーの養成が可能となった。講習会は定期的に開催しており、現在約50名が受講済みでシステムの開発にあたっている。2000年度上期中にほぼ全員のプログラマーに対して教育を行う計画である。

6.2　フレームワークの品質管理

開発したフレームワークを検証するため、設備投資管理システム及び生産管理システムをパイロットシステムとして試行することにした。しかし、開発を始めると昨日まで問題なく動いていたプログラムが突然動かなくなるというトラブルが発生した。原因は、システム開発に必要な新たな機能を既存部品に追加した際の修正ミスであった。プログラムの再利用はコーディング量の削減、生産性の向上というメリットがあるが、一方で再利用率が高い部品ほど品質に問題があった場合の影響が大きく、保守が難しくなる。ニーズに合わせて機能拡張されていく部品の品質を確保するため、次の２つの対策をとった。

(1)　単体テストの徹底

プログラム変更に伴うバグを防ぐために単体テストの精度を上げることにした。過去の経験によればプログラム変更が数回繰り返されると単体テスト仕様書の改訂漏れが発生し、必要なテスト項目を漏れなくテストすることが難しくなる。プログラムの変更に合わせて単体テスト項目の変更が同時にできるよう、プログラムソース先頭のコメント行に特定のキーワードと共にテスト

項目を記述することにした。次に単体テストが漏れなく行われていることを確認するため、次のツールを作成した。ソースファイルから更新日付及びテスト項目抽出し、品質管理ＤＢに登録する。次に、プログラマーが単体テストを行い、結果を入力する。ソースを修正すればファイルの更新日が変更されるため、テストは未実施のステータスに変更される。このような仕組みを確立し、テストが完了していない部品のリリースが防止できるようにした。しかし、単体テストの実施にはコストがかかるため、現在、単体テストの自動化を検討している。

(2)　プログラムソースのバージョン管理

万一、プログラムの変更により不具合が発生した場合、いつ、だれが、何の目的でプログラムを修正したかという変更記録がなければ適切な対応が難しくなる。開発標準ではプログラムソースのコメント部分に変更日、変更者、変更内容を記述するよう指示しているが、バグはプログラマーが影響範囲を正確に捕らえていないために発生することが多く、変更内容の記述に現れてこないことが多い。そこで、すべてのバージョンのソースと変更日時、変更者を自動的に保管するツールを作成した。このツールはＧＵＩ画面からJavaコンパイラーやエディターを起動する簡単なものであるが、コンパイルや編集のタイミングでファイルの日付をチェックし、前回と異なっていればそのファイルを別のエリアに保管するよう改良した。このツールによりプログラムの履歴を完全に追うことができるため、バグ修正された部品が素早くリリースできるようになった。

7．評価と今後の課題

7.1　評価

以上の取り組みによりフレームワークおよび部品を開発し、パイロットシステムを開発することができた。２つのシステムはいずれもプログラム本数約３０本であるが、ユーザー管理、メニュー管理、権限管理の部分はフレームワークをカスタマイズせずにそのまま利用したため、個別に開発したプログラムは設備投資管理システムが２１本、生産管理システムは１７本であった。表3に以前Cold Fusionで開発したシステムと今回開発したシステムのライン数を示す。部品の活用によりHTML出力のためのコーディングが大幅に削減され、平均ライン数がそれぞれ50%, 39%削減できた。設備投資管理システムの方が削減率が高いのはプログラムパターンの適用率が高いためである。

表３　パイロットシステムと従来システムのプログラム・ライン数の比較

シ　ス　テ　ム	開発言語	開発本数	全ライン数	平均ライン数(従来比)
設備投資管理	Java Servlet	21本	6,057	288 (50%)
生　産　管　理	Java Servlet	17本	5,967	351 (61%)
外　注　管　理	Cold Fusion	21本	12,216	581 (100%)

表４に本プロジェクトの目標とパイロットシステムでの成果を示す。表３で示したライン数削減による生産性の向上及びユーザー管理、メニュー管理、権限管理フレームワークの利用により目標以上の開発工数削減、開発納期短縮が実現できた。

表４．目標と成果

	目　　標	成果 (設備投資管理システム)	成果 (生産管理システム)
コーディング量	50%削減	約50%削減（12→6KStep）	約39%削減（10→6KStep）
開　発　工　数	30%削減	約50%削減（3.0→1.5人月）	約44%削減（2.5→1.4ヶ月）
開　発　納　期	30%削減	約50%削減（3.0→1.5人月）	約44%削減（2.5→1.4ヶ月）

(補足) 当社実績ではCold Fusion の生産性は COBOLに比べ約30%高い

7.2　今後の課題

将来の目標は業務毎にフレームワークを構築し、新規機能を追加するだけで新規システムの開発ができるようにすることである。現在、構築したフレームワークを使って購買管理システム、物流管理システム、生産管理システム、営業支援システムを開発しているが、これらのシステムで開発したプログラムをフレームワークにするためには今後発生する機能追加を予想し、差分プログラミングができるようにしておかなければならない。購買管理システムについては購買フレームワークとして外販する予定で開発を進めている。このシステムの開発の中でフレームワークを構成するプログラムの開発手法を確立し、今後開発するシステムがフレームワークとして他のシステム開発で再利用できるようにしたいと考えている。

8．おわりに

他社ではJava言語をCOBOL同様の手続き型言語として使用している例が多く、一般的には画面(HTML出力)とプログラムロジックを分離するためにJSP(Java Server Pages)の技術が推奨されている。しかし、当社ではオブジェクト指向言語のメリットを引き出すため、敢えてJSPを使わずにJavaだけでシステムを構築した。Java Servlet を使ったシステムの構築を検討されている方々に何らかの参考になれば幸いである。