Web アプリケーション開発における Java フレームワークの適用
―はじめての Java へのアプローチ―

さわぐち まさる
澤口　優
[略歴]
1976 年　アラコ株式会社入社
現在　　情報システム部　事務システム室 室長
システム経験年数　25 年

かとう　まき
加藤　真樹
[略歴]
1993 年　アラコ株式会社入社
現在　　情報システム部　事務システム室
システム経験年数　8 年

原稿量
本文　　11,900 字
要約　　1,100 字
図表　　14 枚

＜要　約＞

当社では、90 年代半ばから、情報インフラ整備とともに、従来のホスト集中型から、クライアント／サーバー型のオープン系システムに移行して、業務の効率化を図り、成果をあげてきた。

しかし、当社を取り巻く環境は、ますます厳しくなり、情報化投資の抑制、開発人員の削減が求められるようになってきた。その中で、情報システム部門における使命は、業務改革の推進とさらなる開発期間の短縮を行い、業務のスピードアップを行なうことである。

その使命を担うべく、我々は基幹業務システムの Web アプリケーション開発に取り組み、開発コストの低減、開発期間の短縮、ＴＣＯの削減をめざすことにした。

Web アプリケーションを構築するにあたり、開発言語は実用化の進んでいる Java を採用した。Java の形態としては、サーバー側で動かすサーブレットを採用した。

はじめて Java で開発するにあたり、Java サーブレットの基本型を調査し、実際に生産性を確かめた。

その結果、Web アプリケーションを開発するには、高度なコーディング技術がなくても、開発できるようなフレームワークが必要であること。そして、開発した部品を明確にして再利用できるようにルール化することが重要であることがわかった。

効率的なフレームワークの骨格を調査し、実際の開発におけるフレームワークの構築を検討した。

フレームワークとして、親クラスに Java の高度なコーディングを記述して、子クラスはそれを継承するようにした。その親クラスをベースとしてコーディングすることで、高度なコーディング技術がなくても開発が可能になった。システム開発において、親クラスをベースとして、全社のシステムで共通項目を記述する。これを継承して、各システムでの共通項目を記述する。各機能別処理は、これを継承して開発することでコーディング量が削減でき、開発期間の短縮につながる。

フレームワークを適用するにあたり、開発した部品を再利用できるよう、標準化をはかった。一つは、部品化の単位を明確にすることと、もう一つはプログラムの機能を明確にするための命名規則を作ることである。標準化することで、オブジェクト指向言語の利点を生かし、開発期間の短縮と、プログラムの維持管理工数の低減をはかることができる。

以上の取り組みにより、Web アプリケーション開発においてフレームワークを適用し、基幹業務システムの開発を行った。その結果、プログラム開発の生産性を 2 倍にすることができた。また、今後、ＴＣＯの削減をはかることができると評価している。

今後は、さらに開発体制を強化し、基幹業務システムの Web アプリケーション開発をすすめていきたいと考えている。

目　次

１．はじめに

当社では、90 年代半ばから、情報インフラ整備とともに、従来のホスト集中型から、クライアント／サーバー型のオープン系システムに移行して、業務の効率化を図り、成果をあげてきた。

しかし、当社を取り巻く環境は、ますます厳しくなり、情報化投資の抑制、開発人員の削減が求められるようになってきた。その中で、情報システム部門における使命は、業務改革の推進とさらなる開発期間の短縮を行い、業務のスピードアップを行なうことである。

その使命を担うべく、我々は基幹業務システムの Web アプリケーション開発に取り組み、開発コストの低減、開発期間の短縮、ＴＣＯの削減をめざすことにした。

本論文では、Web アプリケーション開発において Java フレームワークを適用した経験にもとづき、はじめての Java に対して、アプローチした内容を紹介する。

２．Web アプリケーション開発の背景とねらい

２.１ Web アプリケーション開発の背景

当社の位置するトヨタグループでは、従来の部門最適から全社最適、さらにグループ最適にすすんでおり、情報インフラと業務の統合をしていく動きがある。

当社の状態は、クライアント／サーバー型システムが増えるたびに、サーバーやデータベースの分散から、開発コストと運用コストは増大していた。これにより、開発面・運用面ともに、これ以上の拡張は困難な状況にあった。

その対処方法は、基幹業務システムにおいて、従来のクライアント／サーバー型システムの構築方法を変革することであった。

今後の構築方法を考えるにあたり、マクロの観点から考えてみると、近年の情報テクノロジーはインターネット技術を利用した、ｅ－ビジネスへと変化している。当然、当社においても、Web の世界は魅力的であり、ＢtoＢサプライチェーンの構築、また新たなビジネスチャンスを創り出すための手段として活用したいと考えていた。

こうして、これからのシステムは、Web アプリケーションが望ましいであろうという結論に達した。今後、基幹業務システムを構築する上での開発環境とすべく、はじめての本格的な Web 開発に取り組むことにした。

Web アプリケーションの構築において、開発言語は実用化の進んでいる Java を採用した。

オブジェクト指向言語でプログラムを部品化できる Java は少人数での開発生産性を高めるのに最適であり、幅広いプラットフォームで動作する将来性のある点でも決定的であった。

Java の形態としては、サーバー側で動かすサーブレットを採用した。なぜなら、一つには、社内で順次整備してきたパソコンには能力差があり、極力パソコンの能力に依存されないようにすること。二つめには、パソコンやサーバーの維持管理工数（バージョンアップやソフト配布等）を減らして運用コストを低減するためにも、サーバーでプログラムを動かすことが必要

と判断したからである。

2 .2 Web アプリケーション開発のねらい

Web アプリケーションを Java サーブレットで開発する上でのねらいは次のとおりである。

(１)システム環境の統合によるコスト低減
(２)Web アプリケーション開発によるプログラム開発の生産性向上
　　→目標値： ２倍

(１)は運用面でのねらいであり、分散しているサーバー、データベース等のシステム環境を統合して、導入コストとランニングコストを低減する。
(２)は開発面でのねらいとして、Web アプリケーション開発により、少人数での短期開発と低コストを可能にする。これにより、プログラム開発の生産性を２倍にすることを目標とした。

３．Web アプリケーション開発検討

3 .1 Web アプリケーション開発要件

Web アプリケーション開発要件として次の２点をあげた。

(１) 情報のリアルタイム性
(２) 全社クライアント数 2,000 台に対してのレスポンスの保証

これらは社内の基幹業務システムすべてにあてはまる要件である。
(１)について、情報のリアルタイム性の条件を満たすためには基幹データベースとの連携が必要になる。
(２)について、クライアント数 2,000 台に対してのレスポンスを保証するために、サーブレットが動くサーバーの能力はもちろん、データベースの設計も考慮しなければならない。
以上をふまえ、アプリケーション環境デザインを検討した。

3 .2 Web アプリケーション環境の選定

Web アプリケーション環境を選定するにあたり、検討した選定順序を図１に示す。
選定する順序は、第１がデータベースであり、重要である。今後、どうしたいのかを十分検討したうえで、既存資産との共存を考えるため、時間をかける必要がある。データベースが決定すると、２番めに、Web アプリケーション実行環境を検討する。これはデータベースとの親和性が第１条件となる。３番めに、それが稼動するアプリケーションサーバーを検討する。これも、データベースおよび、Web アプリケーション実行環境との親和性が第１条件となる。

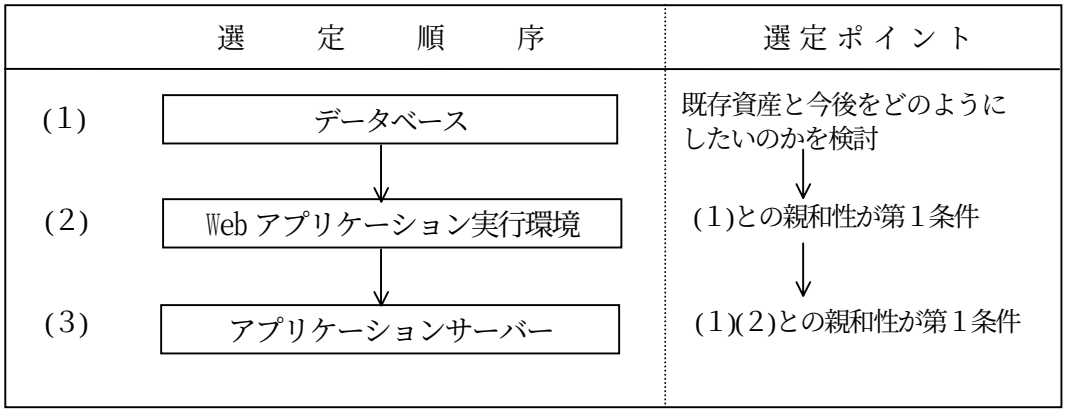

図１．Web アプリケーション環境の選定順序

当社において、選定順序(１)のデータベースの選定をする際、既存のメインフレームであるＯＳ／３９０（以下、ホストＤＢ２）を使用するのか、あるいはＮＴやＵＮＩＸにもたせるのかを検討した。その際の、相対評価を表１に示す。

表１．データベース選定のための相対評価

	性能	コスト	基幹業務連携の容易性	安定性	レスポンス	合計点
ホストＤＢ２	○	○	○	○	×	8
ＵＮＩＸ	△	×	△	△	○	5
ＮＴ	×	△	△	×	△	3

相対評価 ： ○…２点、△…１点、×…０点

相対評価の合計点はホストＤＢ２が８点で最高点となった。

表の相対評価の中で、コストとレスポンスに関して考察する。

はじめにコストについて、ホストＤＢ２が相対評価○となっているが、これは、ホストＤＢ２が既存資産であるのに対して、ＵＮＩＸとＮＴは新規購入するためにこのような評価をした。

次にレスポンスについてであるが、ＵＮＩＸとＮＴの場合はデータベースをアプリケーションサーバーと同じローカルにもつことになる。これに対して、ホストＤＢ２の場合はデータベースがリモートアクセス(注1)になることを考慮してこのように評価した。

検討した結果、相対評価の合計点の高いホストＤＢ２に決定した。レスポンスに関しては心配もあったが、ＳＱＬ文を発行して返ってくるレスポンスは現在稼動しているシステムで実証していることから、アプリケーションの設計方法で対応可能と考えた。

注1：トヨタグループボデーメーカーでホストコンピュータの統合を推進中であり、当社ホストＤＢ２はトヨタにある。

次に選定順序(２)の Web アプリケーション実行環境を検討する際、ホストＤＢ２との親和性をまず第１の条件として、条件を満たした WebSphere を選択した。

そして、選定順序(３)において、アプリケーションサーバーは、ＮＴとＵＮＩＸのどちらかに決定することになった。最終的にＵＮＩＸを採用した理由はＮＴサーバーと比較しての安定性とクライアント数全社 2,000 台に耐えうる能力が必要だったためである。

以上の結果を総合して、Web アプリケーション開発要件を満たした環境を選定した。

図２に、選定した Web アプリケーション環境を示す。

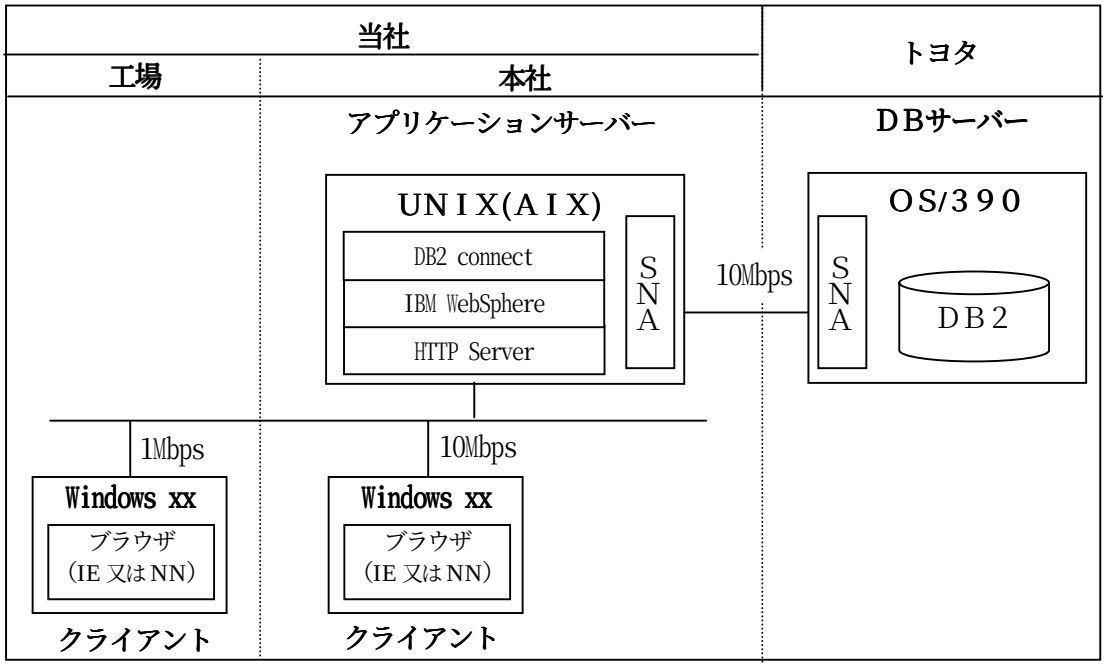

図２．当社の Web アプリケーション環境

３．３システム開発要件

基幹業務システムにおける Web アプリケーションシステムを開発する上での要件を具体化した。最初のシステム開発として、事務・技術部門の勤務日報システム(勤怠と工数の入力システム) を開発対象とした。本システムは、クライアント数 1,500 台で、終業時あるいは翌日の始業時に出退社・労働時間・残業等の勤怠情報の入力、および業務 (プロジェクト) ごとの工数を入力するシステムである。入力・承認したデータは、給与支払いシステム、原価管理システム等に連動する。

具体的な開発要件は次のとおりである。

(１) 同時アクセス集中に耐えうるシステム設計 (同時接続数の想定 300 台)
(２) メインの勤怠入力画面でのレスポンスの保証
　→目標値：１クリックしたときのレスポンス５秒以内

（1）の同時アクセス集中について、特に考慮が必要だった。全社基幹業務システムであるため、ユーザー全員が毎日使用するシステムであること。入力タイミングが入退社時の前後になるということから同時接続が予想されること。以上のことから前もって、同時アクセス集中時の対策をすることで、ユーザーにとって、ストレスのないレスポンスを保証する必要性があった。

(2)のレスポンスの目標値については、インターネットの閲覧とは違い、基幹業務での入力ということで、ユーザーがストレスを感じない程度の５秒以内を目標値とした。

これらの目標を確実にクリアするために、あらかじめシステムの特性を分析した上で、システム設計を実施した。

３．４ システム設計

３．４．１ レスポンス悪化の想定要因

システム設計を行なうにあたり、レスポンス悪化の想定要因を探るために、システムの特性を抽出した。

勤務日報の情報は、給与支払い等の処理へつなぐため、稼働日１日中に、ユーザーは先月の自分の勤怠・工数のデータを確定しなければならない。勤務情報に、間違いがあった場合は修正し、所属長が再度承認を行なう。当然ながら、今月の計画も新たに入力しなければならない。システムの特性から、稼働日１日はアクセス数が増えることが容易に予想できた。

稼働日１日の入力項目と機能について、図３のように図解して検討した。

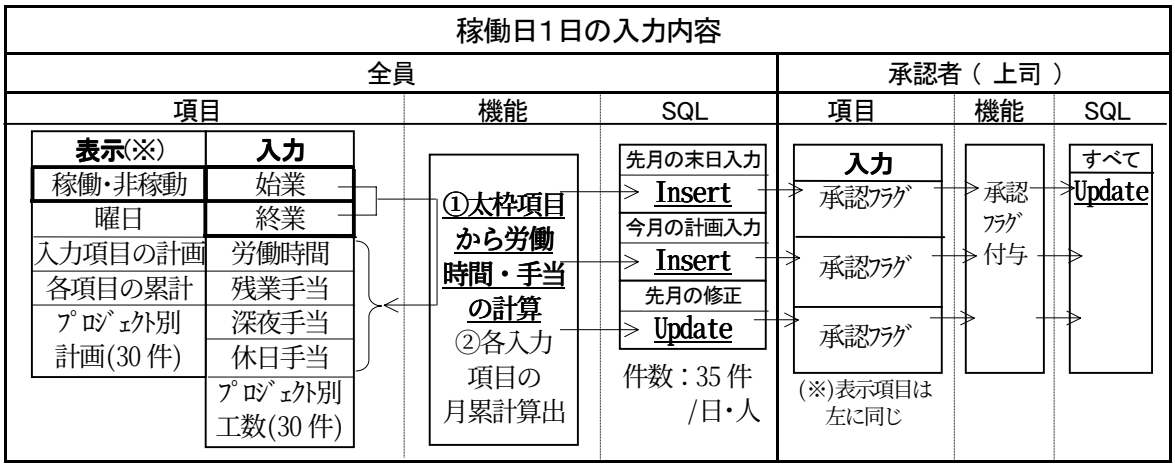

図３．稼働日１日の入力項目と機能

この図から、レスポンス悪化の要因として気づいた点は、

(1) 同時アクセス集中時（稼働日１日）に Insert 文が集中する。
(2) 表示･入力項目が多く、手当の計算が複雑である。
ということであった。

（1）について、Insert しなければならないデータは、工数データと勤怠データを合わせると 35 件／日･人 になる。ユーザー全員は、これを稼動日１日に Insert するのである。同時アクセス数が 300 とすると、35 件×300 で 10,500 件の Insert が同時に行なわれる可能性があった。これを回避するために、時差入力するなどして運用するには無理があると考えた。

（2）の表示項目については、勤怠には残業、手当、などのほか、曜日、稼働日情報なども表示しなければならず、項目を必要以上に減らすことはできない。

以上、レスポンス悪化の想定要因をふまえた上で、システム設計に着手した。

３．４．２ システム設計

同時アクセス集中時に、レスポンスを悪化しないためにはどうしたらよいか。さらに、表示項目を減らさずに入力をシンプルにするにはどうしたらよいか検討した。

そこで、必ずしも Java でリアルタイムに行なわなくてもよいロジックは、ホストであらかじめ処理しておくという方式を考えた。さらに、表示項目については、減らすことはできなくとも、月単位の入力を１日単位の入力にするなど、１画面に対しての項目は減らすことができると考えた。具体的には、次のとおりである。

（1）翌月データの作成はホストのバッチ処理で空データを作成しておく。
（2）曜日、稼働日などの表示項目を区分化してあらかじめデータにもたせる。
（3）１画面に対しての項目を少なくする。

（1）について、あらかじめ作っておいた空データを Update すれば稼動日１日の大量データ Insert を避けることができる。なぜなら、Update は Insert にくらべてレスポンスが速いためである。

（2）について、空データを作成する際に、曜日や稼動日を区分化し、すべて情報として付与してしまうことで、Java ではその区分を識別して表示するだけにした。こうして、入力データに稼働日情報をあらかじめ付与することで、稼働日か非稼働日かどうかによる手当の計算が、簡単に早くできるような設計にした。

レスポンス面だけでなく、開発期間の短縮からもこの方法でよかったと思っている。なぜなら、Java 言語は、簡単なほうではなく、ホストの開発言語と比較しても習得が難しいからだ。難しいということは、ロジックが複雑になるほど、開発期間が長引くことにつながるからである。

このように、必ずしもリアルタイムで行なわなくてもよい処理は、設計段階で思い切って別の方法で行なうよう設計することが大切である。

一方、ホストのバッチ処理の開発は、短期間で開発できた(400step で 16ｈ／本)。

（3）の画面設計については、Visual Basic の画面のように１画面の機能が多くならないようにシンプルさを考慮した。

以上、システムの設計が完了し、Java の開発に着手した。

４．Web アプリケーション開発におけるフレームワークの適用

４．１ サーブレットの基本型

はじめて Java で開発するにあたり、Java サーブレットの基本的な開発方法を調査した。一般的に Java サーブレットの基本型として扱われているものがＭＶＣ(モデルビューコントローラ)モデルである(1)。

Ｍ：モデル(業務ロジック)
Ｖ：ビュー(見栄え)
Ｃ：コントローラ(サーブレット)

(１) モデル(Bean)
・実際の業務ロジックを記述する。
(２) ビュー(JSP)
・Bean の処理結果をもとに HTML ファイルを作成してクライアントに返す。
(３) コントローラ(Servlet)
・要求をコントロールする。

以上３つの役目を切り分けた設計をすると、サーバー側の実行環境がたいへんシンプルな形になり、再利用性が高くなると言われている。この理由から、ＭＶＣの形をとることにした。

一方、サーブレットの開発は JSP を使用しなくてもできる。しかしながら、JSP を使用することにした理由は、業務ロジックと切り離せるのが利点だと思ったからである。JSP を使用して設計することにより、ユーザーからの画面仕様の変更・機能追加の要望に対して、JSP のみ修正し、迅速な対応ができること、さらにテスト工数の低減が期待できる。

４．２ Java の生産性の実証

以上のとおり、Java での開発をＭＶＣと決め、実際に生産性が向上するかを確かめた。ＭＶＣの基本型は、ひな型生成ツールで簡単にひな型を作成することができるので、勤務日報システムのメイン画面を試作することにした。

やり方は、

(１)アクセスする SQL 文の条件をパラメータ式で入力して、ツールでひな型を作成する。
(２)作成したひな型に業務ロジックを追加していく。

という形をとった。

すると、生産性があがるどころか、メイン画面一つ作るだけで１ヶ月以上かかってしまった。そのときのプログラムのコーディング量（ step 数 ）を表２に示す。

表２．ひな型生成ツールをもちいたコーディング量(１画面)

機能	コーディング量	開発工数
勤怠表示機能	1,500 step	１人月
勤怠入力機能	3,500 step	

開発が長びいた原因としては、

(1) ひな型サンプルのソースは難解であるため、ロジックを追加していくのが困難であった。
(2) ひな型サンプルプログラムを各機能毎に作成したため部品化できずに、ロジックが重複した。

（１)について、Java の開発経験が２～３ヶ月であり高度なコーディング技術がなかったため、ソースの解読だけで時間がかかってしまった。

（２)について、SQL 文の単位で作成したため、同じロジックがいろいろな場所にでてきてしまった。これが原因で、無駄の多いプログラムとなってしまったためにコーディング量が増えて、開発期間が長期化した。

以上のとおり、Java の生産性を確かめた結果、

(1) プログラマーに高度なコーディング技術がないと、開発に時間がかかり、生産性はあがるどころか逆に低下してしまう。
(2) プログラマーがプログラムを部品化したことを明確にしないと、再利用できずにロジックが重複してしまう。

ということがわかった。

そして、開発方法については、ひな型生成ツールはコーディングをせずにプログラムが作成できるため、情報公開システム等のように単体で動かすなら便利で有用である。しかし、一度生成したプログラムにロジックを追加していくのは困難であり開発工数がかかるということがわかった。当社のニーズは、基幹業務システムの開発であり、業務ロジックは必ず発生することから、この方法は不向きであると判断し、使用せずに開発を行なうことにした。

以上により、開発方法は、将来性を考えて、オープンな環境で開発できるということから、フリーソフトのエディタでコーディングし、JDK でコンパイルするという形をとることにした。

４．３ フレームワークを適用するための必要事項

以上のとおり、実際に Java で生産性を確かめてみて、Web アプリケーションを開発するには、

（１）高度なコーディング技術がなくても、開発できるようなフレームワークが必要である。
（２）開発した部品を明確にして再利用できるようにルール化する。

以上の２つが社内での開発はもちろん、アウトソーシングするにしても大切であり、必要なことであると考えた。

4. 4 フレームワークの骨格

フレームワークの骨格がきちんとしていると、Java の生産性と再利用性を高めることができる。これについて調査した結果、業務ロジックを記述するMの部分は部品化することで、プログラムの再利用性が高まることがわかった(2)。図４に、ＭＶＣのそれぞれの役割分担とコンポーネントの動きを示す。

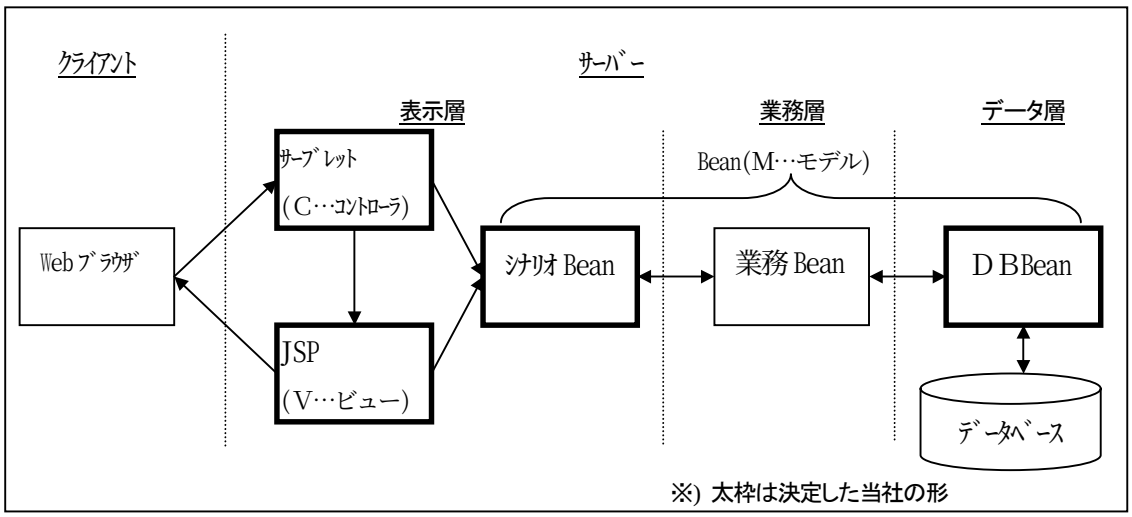

図４．ＭＶＣの役割分担とコンポーネントの動き

ＭＶＣのなかで特にM（以下、Bean）の業務ロジックの部分について、Bean の中でも役割があり、ＤＢアクセス部分はＤＢBean で行なうこと、業務ロジックは業務 Bean で行なうことが理解できた。

当社の場合は業務 Bean を独立するほど業務ロジックがなかったので、今回はサーブレット・JSP・シナリオ Bean・ＤＢBean の形をとった。

４．５ フレームワークの構築

４．５．１ フレームワークの理想型

フレームワークの骨格が決まったところで、実際のフレームワークはどのように作ったらよいか検討した。

Java は親クラスを継承して子クラスを作成することができる。親クラスには共通の仕様を記述して、それを継承してそれぞれ子クラスを作成すれば、コーディングが少なくてすむため効率的である。こうして親クラス(Base ｸﾗｽ)を継承して開発する例を図５に示す。(2)

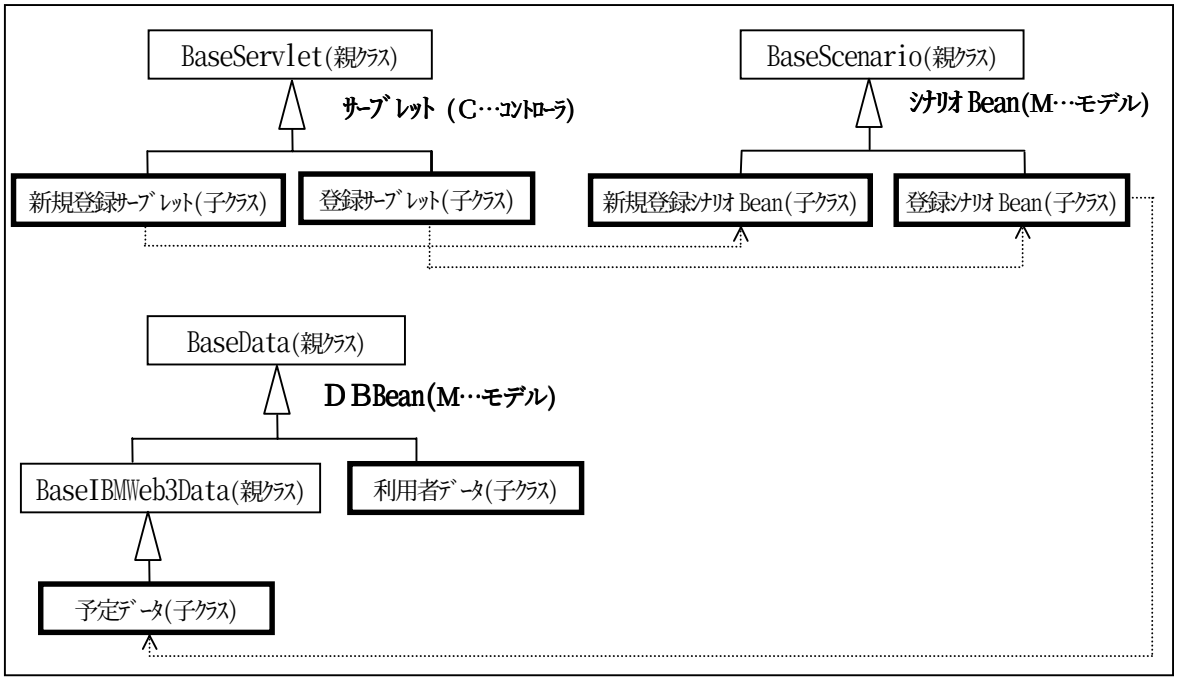

図５．親クラス(Base)を継承しての開発例

親クラスにJavaの高度なコーディングを記述して、子クラスはそれを継承するようにする。このように開発すると、プログラマーは太枠で示す子クラスのみの開発になる。親クラスの変数はパラメータをセットするようにしておけば、難しいコーディングが理解できなくてもプログラム開発が可能になる。こうして、プログラマーは差分のみを記述すればよいことになるため、プログラム開発の生産性が高まるのである。

４．５．２ フレームワークの構築

実際のシステム開発において、生産性をあげるためには、シナリオ Bean において、システムでの共通の項目(機能)を洗い出して部品化する必要がある。

全社システムであるいは勤務日報システムで共通項目は何か。全社のシステムで共通項目は、Web アプリケーションにログインするために必要な情報であった。この情報を継承すると、プログラムでの記述の重複をさけることができ、全社システムで共通項目が増えた場合もメンテナンスが容易になる。それに加えて、勤務日報システムでの共通項目を、継承するようにシナリオ Bean を設計する。以上の結果、シナリオ Bean は図６のようなプログラムはツリー構造になった。

図６のように、全社共通 BaseScenario を作成しておくと、実際の開発では、太枠の部分を開発することになる。勤務日報 BaseScenario には勤務日報システムでの共通部分をコーディングする。そして、各機能別に、新規登録シナリオ Bean あるいは登録シナリオ Bean 等は勤務日報 BaseScenario を継承して開発すると、コーディング量が少量ですむことになる。

こうしてフレームワークが構築できたところで、次に開発していくうえでのルールを考えた。

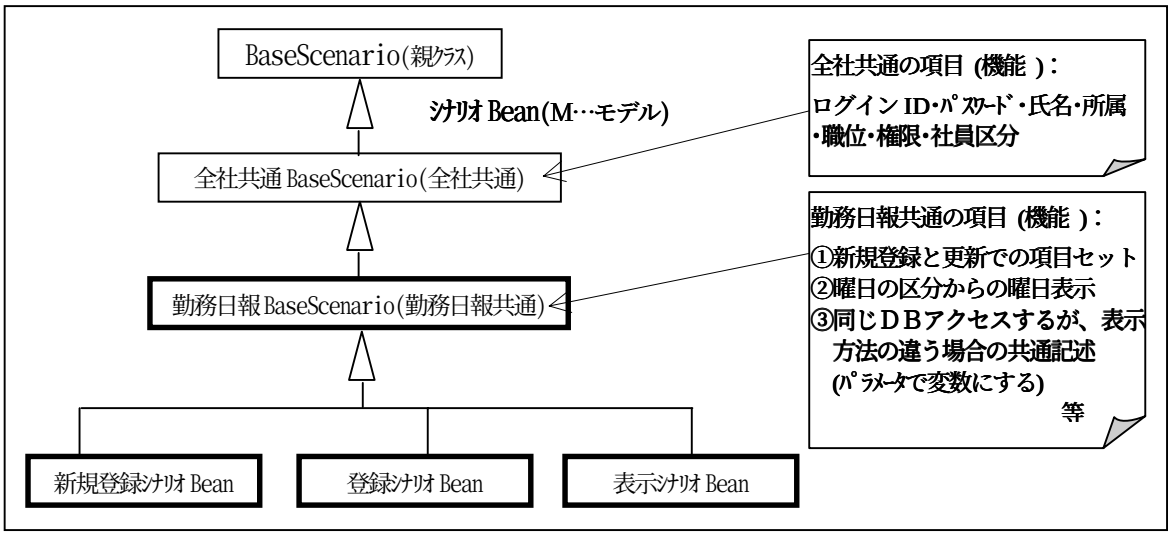

図６．当社におけるシナリオ Bean の形

４．６開発のルール

今後、さまざまなシステムを開発していくと、プログラムの量が増えていく。それにつれて、どこに何が書いてあり、どのシステムに使用しているのか煩雑になりがちである。煩雑になってから後で統一するのは困難なので、私が考える最低二つの開発のルールを以下に述べる。

(１) 業務ロジックの部品化の単位を明確にする。
(２) プログラムの機能を明確にするため、プログラムの命名規則を作る。

(１)については、オブジェクト指向といえども、開発の単位はシステム単位であることが多い。全社のシステムを一から作り直すなら別として、部品化の単位はシステムの単位であるのが安全だと思う。なぜなら仕様変更が入り、ベース(親クラス)となるプログラムを直すたびに、全てのシステムが関係していると、全てのシステムについて毎回テストしなければならなくなるからである。

したがって、全社でベースとなるプログラムは不変性があり、統一性のある部分を部品化し、あとはシステム毎に業務ロジックを部品化することが有用であると思う。システムをまたいで業務ロジックを部品化して継承するのは理想ではあるが、仕様の段階で長い時間を要する。開発期間の短縮を求めるなら、システム単位で割り切ることをすすめたい。

当社の場合、業務ロジックはシステム内でとどめているが、ＤＢBean に関しては、アクセスするテーブルに対してＤＢBean を１つだけ作成して、各システムで共有している。
特にＤＢBean の場合は、同じ SQL 文があちこちで発生しないためにも共有するのが望ましい。

(２)については、ＭＶＣは１：１：１で作成することから、表３のようにどのコンポーネントと対応しているのかを明確にする表を作成する必要がある(2)。

表３．コンポーネント対応表

No	画面名称	JSP	ｻｰﾌﾞﾚｯﾄ	ｼﾅﾘｵ Bean
1	画面名称１	Screen1.jsp	Screen1	Screen1Bean
2	画面名称２	Screen2.jsp	Screen2	Screen2Bean

オブジェクト指向であるプログラムを修正する際には、どこに何が書いてあるかを明確にしておく必要がある。なぜなら、部品化されているプログラムの修正箇所を探すだけで工数がかかるからである。そこで、たかがプログラムの名前といえども、一目見て機能がわかるように命名しておけばプログラムも探しやすい。

表３のコンポーネント対応表につけ加えて、表４に当社での命名規則を示す。

表４．当社における命名規則

ｼｽﾃﾑ名	ﾒﾆｭｰ番号	機能連番１	機能
Kinmu	01	1	Select

No	画面名称	JSP	ｻｰﾌﾞﾚｯﾄ	ｼﾅﾘｵ Bean
1	勤怠表示画面１	Kinmu011Select.jsp	Kinmu011SelectServlet.java	Kinmu011SelectBean.java
2	勤怠表示画面２	Kinmu012Select.jsp	Kinmu012SelectServlet.java	Kinmu012SelectBean.java

コンパイル時の利便性と管理のしやすさを考えて、頭はシステム名で統一してある。

修正や変更が入るのはシステムの単位か、メニューボタンの単位である。この命名規則であれば、システムあるいはメニューボタンの単位で１回(１行)でコンパイルできる。

例えば勤務日報システムの場合、全てプログラムの頭に Kinmu がついている。JDK でコンパイルする際に、頭文字が同じであるため、Kinmu*.java でコンパイルできる。さらに、勤務日報システムのメニュー①の仕様変更があった場合は Kinmu01*.java とコンパイルすればよい。

また、各プログラムの機能を識別するのには Select、Update、Insert とプログラムの名称に機能を入れてしまえばプログラムは何が記述してあるのか中を見なくてもわかるようになる。

一方、ＤＢBean は共有するので、サーブレット：JSP に対して１：１：１にはならないが、当社ではアクセスするテーブル名と同じ名前のＤＢBean を作成するようにしている。こうすることで、一目でどのテーブルにアクセスしているかがわかり賢明である。

５．Web アプリケーション開発の実践

５．１フレームワークの適用事例

フレームワークに当てはめて勤務日報システムと、カフェテリアプラン(全社福利厚生)システムを開発した結果、開発は順調に進んだ。表５に、システム開発実績を示す。

表５．システム開発実績

システム名	ﾒﾆｭｰ数	画面数(JSP)	ｻｰﾌﾞﾚｯﾄ		Bean						ﾌﾟﾛｸﾞﾗﾑ開発工数
					ｼﾅﾘｵ Bean (BaseScenario)		ｼﾅﾘｵ Bean (子ｸﾗｽ)		ＤＢBean		
			本数	総 step 数	本数	総 step 数	本数	総 step 数	本数	総 step 数	
勤務日報ｼｽﾃﾑ	14	56	56	3,900 (平均 70)	4	7,000 (平均1,700)	56	8,400 (平均 150)	8	6,000	2 人月
ｶﾌｪﾃﾘｱﾌﾟﾗﾝｼｽﾃﾑ	7	28	28	2,000 (平均 70)	2	3,000 (平均1,500)	28	4,200 (平均 150)	10	3,000	1 人月

5. 2 JavaMail への応用

今回の勤務日報システムには、ワークフローの機能として休務届申請が含まれている。休務届は休みをとる申請をし、所属長がそれをうけて承認するという機能である。休みをとる申請をする際に所属長にメールが送られる機能を開発したかったため、すでにメールツールとしているノーツとの連携を考えた。JavaMail を使用して、フレームワークにならって開発したが主な工夫点は次ぎのとおりである。

(１) メール送信に必要な機能を部品化する。
(２) 送信文書をテンプレート形式でサーバー上にもち、プログラムの記述の中ではパラメータをセットする。

(１)については、メール送信に必要な機能を部品化することで、簡単にコーディングできるようにした。

(２)については、送信文書のフォーマットをテンプレート形式でもつことで、今後 Web 上でのワークフローシステムが増えても、新しいテンプレートを追加して、同じやりかたで開発できるように工夫した。こうしてメール送信機能は拡張性の高い機能とすることができた。

６．評価と今後の課題

６．１評価

６．１．１ 開発面での評価

以上のとおり、Web アプリケーション開発においてフレームワークを適用したことで、次のように成果をあげることができた。

フレームワークを使用した場合と、未使用の場合の想定値とのコーディング量の比較を表６に示す。フレームワーク使用の実績値については、５章の表５から、サーブレット、Bean の合計 step 数を算出した。フレームワーク未使用の想定値は４章の表２より 5,000step／画面として、×14 画面(メニュー)で算出した。フレームワークを使用することで、コーディング量を大幅に削減することができた。

表６．フレームワーク使用の有無によるコーディング量の比較

システム	**ﾌﾚｰﾑﾜｰｸ使用の実績値**	ﾌﾚｰﾑﾜｰｸ未使用の想定値	**削減量**
勤務日報システム	**25,300 step**	70,000 step	**64%**

次に従来の開発との比較を表７に示す。同じ機能をもった勤務日報システムを、現業部門用に４年前、従来のクライアント／サーバー型システムで構築した。この表は、その際の開発工数との比較である。プログラム開発の生産性を２倍にすることができ、当初の目標を達成した。これにより、開発期間の短縮をはかることができた。

表７．開発工数の比較

システム	**Web での開発**	従来方法での開発	**生産性**
勤務日報システム	**２人月**	４人月	**２倍**

これらは、Java の開発経験年数が２～３ヶ月(基幹業務システム開発経験年数２年・７年)といったプログラマーでの実績値である。

今回構築したフレームワークをベースに、今後の基幹業務システムの開発を Web アプリケーションで推進できると、評価している。

６．１．２ 運用面での評価

運用面での評価は以下のとおりである。

(１) システム環境の統合による、サーバー・データベースのコスト低減
　　(バックアップ、バージョンアップ等の運用コスト含む)
(２) システム毎のパソコンへのソフトの導入や、バージョン管理の不要
(３) パソコン環境の違いによるトラブルの解消
(４) アプリケーションの変更のたびに発生するソフトの配布工数の解消

以上のことから、大幅にＴＣＯの削減をはかることができると、評価している。

６．１．３ ユーザーの声

ユーザーの評価はよく、次の点で満足している。

（1）レスポンス面
（2）シングルログインで各システムにアクセスできる
（3）操作の統一性
（4）データのリアルタイム性

クライアント数1,500台に対して画面表示・データ更新のレスポンスもよい。勤務日報システムでのレスポンス実測値を表８に示す。

表８．レスポンス実測値(同時接続数300台)

	本社(10M)	工場(1M)
勤怠入力(ﾒｲﾝ画面)	３秒	３秒
勤怠１ヶ月表示	２秒	２秒

( )内はﾈｯﾄﾜｰｸ回線値

当初、システム開発要件であった、同時アクセス集中時の勤怠入力画面でのレスポンスが、5秒以内で保証できた。

６．２今後の課題

Web アプリケーションを開発していく上で考える課題が２点ある。

６．２．１仕様書の管理

オブジェクト指向では今までの仕様書と考えを変えなければならない。当社の今までの開発では、仕様書はシステムを構築したというその時点での歴史にすぎなかった。もし、仕様書が整備されていなかったとしても、直さなければならないプログラムのロジックだけを理解できれば仕様変更に対応できた。

しかし、オブジェクト指向の場合、今までに開発した中でどんな部品を作成したか、ということを理解することが、プログラムを共有する上で大切である。システムに仕様変更が入った場合、どこをどうなおせばよいのか、末端のプログラムから溯ってシステムを探るのは工数がかかる。仕様変更・仕様追加にも迅速に対応するためには、データベース化した仕様書が必要だと考える。

６．２．２ 開発体制の強化

現在、当社で Java が開発できる人員を２名確保できたが、各ＳＥ・プログラマーへの横展開を早急に行なうことが、基幹業務の開発をすすめていくうえで急務である。そのために、Java の知識だけでなく、フレームワークの必要性、仕様書の書き方、ルールの徹底を理解してもら

うことが大切である。そして、社内での開発体制を強化し、アウトソーシングする際にも、社内で詳細仕様まで設計できるような体制にすることが、無駄な部品を作って生産性が悪くならないようにするために必要なことだと考えている。

７．おわりに

今回、Web アプリケーションを開発するにあたり、Java フレームワークを適用し、短期間で開発することができた。はじめての Java で、しかも基幹業務を開発したというのは貴重な経験であり、思いのほか成果をあげることができたと自負している。今後、このフレームワークをベースに、基幹業務システムの開発を Web アプリケーションで推進していきたいと考えている。

開発する上で、標準化はオブジェクト指向言語の利点を生かすために重要なことであり、開発期間の短縮と、プログラムの維持管理工数の低減につながる。

これから Java での開発を検討している方々、あるいはすでに Java の開発に着手している方々に、何からの参考になれば幸いに思う。

参考文献：

(１) 藤田一郎：「Java がわかる」、技術評論社、1999 年 12 月 25 日、103 頁
(２) A.Saimi,T.syomura,H.Suganuma,I.Ishida.："Presentation layer framework of web application systems with server-side Java technology"、In Proc.COMPSAC、2000、473-478 頁