Web アプリケーション フレームワーク 概要

Sun Java™ Studio Enterprise 7 2004Q4

Sun Microsystems, Inc.
www.sun.com

Part No. 819-1288-10
2004 年 12 月, Revision A

原典: *Web Application Framework Overview*
Part No: 819-0726-10
Revision A

目次

はじめに

このガイドでは、Web アプリケーションフレームワークの紹介、その内容と仕組み、他の Web アプリケーションフレームワークとの相違点を説明します。

お読みになる前に

このマニュアルを読み始める前に、サーブレットや JavaServlet™ ページ (JSP ページ) などの既存の J2EE Web テクノロジを利用した Web アプリケーションの構築で用いられている概念を理解しておくことを推奨します。

詳しい情報は、以下のリソースから得ることができます。

- Java 2 Platform, Enterprise Edition Specification
  `http://java.sun.com/j2ee/download.html#platformspec`
- J2EE Tutorial
  `http://java.sun.com/j2ee/tutorial`
- Java Servlet Specification バージョン 2.3
  `http://java.sun.com/products/servlet/download.html#specs`
- JavaServer Pages Specification バージョン 1.2
  `http://java.sun.com/products/jsp/download.html#specs`

注 – Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスの利用あるいはそれらのものを信頼することによって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

マニュアルの構成

第 1 章、1 ページの「Web アプリケーションフレームワークの概要」では、Web アプリケーションフレームワークの概要を説明しています。

第 2 章、27 ページの「Web アプリケーションフレームワークの設計とアーキテクチャに関してよく受ける質問 (FAQ)」では、Web アプリケーションフレームワークを初めてご使用になる方からよく受ける、設計とアーキテクチャに関するいくつかの質問に対する回答をまとめています。

書体と記号について

書体または記号*	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例。	.login ファイルを編集します。 ls -a を実行します。 % You have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表します。	% **su** Password:
AaBbCc123 または**ゴシック**	コマンド行の可変部分。実際の名前や値と置き換えてください。	rm *filename* と入力します。 rm **ファイル名** と入力します。
『 』	参照する書名を示します。	『Solaris ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照。 この操作ができるのは「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅をこえる場合に、継続を示します。	% **grep '^#define \ XV_VERSION_STRING'**

* 使用しているブラウザにより、これら設定と異なって表示される場合があります。

関連マニュアル

Java Studio Enterprise のマニュアルとしては、Acrobat Reader (PDF) 形式のマニュアル、チュートリアルと、HTML 形式のリリースノート、オンラインヘルプ、チュートリアルが提供されています。

オンラインで入手可能なマニュアル

ここで紹介しているマニュアルは、docs.sun.com℠ Web サイトおよび Sun Java Studio Enterprise Developers Source ポータルサイト (http://developers.sun.com/jsenterprise) のドキュメントリンクから入手できます。

docs.sun.com Web サイト (http://docs.sun.com) では、インターネットで Sun のマニュアルを参照、印刷、購入することができます。

- 『Sun Java Studio Enterprise 7 2004Q4 リリースノート』- Part No. 819-1302-10

  最新のリリースの変更点や技術的な注意事項を説明しています。

- 『Sun Java Studio Enterprise 7 インストールガイド (PDF 形式)』 - Part No. 819-1300-10

  サポートしている各プラットフォームへの Sun Java Studio Enterprise 7 統合開発環境 (IDE) のインストール方法を説明しています。システム要件やアップグレード方法、サーバー情報、コマンド行スイッチ、インストールされるサブディレクトリ、データベースの統合、アップデートセンターの使用方法などの関連情報も記載されています。

- 『J2EE アプリケーションのプログラミング』- Part No. 819-1298-10

  EJB モジュールや Web モジュールを J2EE にアセンブルする方法を説明しています。また、J2EE アプリケーションの配備や実行についても説明しています。

- Web アプリケーションフレームワークのマニュアル (PDF 形式)
  - 『Web アプリケーションフレームワーク コンポーネント作成ガイド』 - Part No. 819-1284-10

    Web アプリケーションフレームワークのコンポーネントアーキテクチャと新しいコンポーネントの設計、作成、配布工程を説明しています。

  - 『Web アプリケーションフレームワーク コンポーネントリファレンスガイド』 - Part No. 819-1286-10

    Web アプリケーションフレームワークライブラリに提供されているコンポーネントを説明しています。

  - 『Web アプリケーションフレームワーク 概要』- Part No. 819-1288-10

    Web アプリケーションフレームワークとその位置づけ、仕組み、他のアプリケーションフレームワークと異なる点を説明しています。

  - 『Web アプリケーションフレームワーク チュートリアル』 - Part No. 819-1290-10

    Web アプリケーションフレームワークを使用して Web アプリケーションを構築する際の仕組みとその手法を紹介しています。

- 『Web アプリケーションフレームワーク 開発ガイド』 - Part No. 819-1292-10

  Web アプリケーションフレームワークを使用し、開発するアプリケーションの構成要素として使用可能なアプリケーションコンポーネントの作成および使用の手順と、そのアプリケーションを大部分の J2EE コンテナに配備する方法を説明しています。

- 『Web アプリケーションフレームワーク IDE ガイド』- Part No. 819-1294-10

  Sun Java Studio Enterprise 7 2004Q4 IDE の各部の概要、および Web アプリケーションフレームワークアプリケーションを開発するためのビジュアルツールの使用方法を重点的に説明しています。

- 『Web アプリケーションフレームワーク タグライブラリリファレンス』 - Part No. 819-1296-10

  Web アプリケーションフレームワークのタグライブラリを簡単に紹介し、タグライブラリに提供されているタグに対する包括的な参照を示しています。

チュートリアル

Sun Java Studio Enterprise 7 には、IDE の機能を理解する手助けとなるチュートリアルがいくつか用意されています。これらのチュートリアルにある技術、およびコード例は、そのまま、または編集を加えて、実際のアプリケーションの開発に利用することができます。すべてのチュートリアルで、Sun Java System Application Server への配備例が紹介されています。

チュートリアルは、すべて Developers Source ポータルのリンク「Tutorials & Code Camps」から利用可能です。IDE で「ヘルプ」>「コードサンプルとチュートリアル」>「概要」を選択すると、このサイトにアクセスできます。

- 「クイックスタートガイド」は、Sun Java Studio IDE の紹介をしています。チュートリアルは、Sun Java Studio を初めてご使用になる方や、特定の機能について早く知りたい場合は、このガイドから始めてください。これらのチュートリアルは、単純な Web アプリケーションや J2EE アプリケーションの開発方法、Web サービスの生成方法を説明しています。また、UML モデリング、リファクタリングの導入方法についても説明しています。ガイドを終えるための所要時間は数分です。
- 「チュートリアル」は、Sun Java Studio IDE の特定の 1 つの機能に焦点を当てています。ある機能の詳細に関心がある場合は、これらを実行してみてください。例で説明している機能によって、初めからアプリケーションを構築する場合と、提供されたソースファイルを使用して構築する場合があります。チュートリアルは 1 時間以内で完成できます。
- 「概要ビデオ」は、技術の説明がビデオで提供されています。IDE の視覚的な概要や、特定の機能の詳細説明を見ることができます。概要ビデオにかかる時間は数分です。概要ビデオは、任意の個所で開始、終了することもできます。

オンラインヘルプ

Sun Java Studio Enterprise 7 IDE には、オンラインヘルプが用意されています。ヘルプキー (Microsoft Windows 環境では F1 キー、Solaris オペレーティング環境では Help キー) を押すか、「ヘルプ」>「ヘルプ (すべて)」を選択して開くことができます。ヘルプの項目と検索機能が表示されます。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは以下の表に示す場所から参照することができます。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアルとチュートリアル	形式: HTML 場所: `http://docs.sun.com`
チュートリアル	形式: HTML 場所: Developers Source ポータル (`http://developers.sun.com/jsenterprise`) のリンク「Examples & Code Camps」
リリースノート	形式: HTML 場所: `http://docs.sun.com`

第1章

Web アプリケーションフレームワークの概要

この章では、以下の節に分けて、Web アプリケーションフレームワークの概要を説明します。

- はじめに : Web アプリケーション構築が抱える難題
- Web アプリケーションフレームワークとは
- Web アプリケーションフレームワークの仕組み
- Web アプリケーションフレームワークと他の J2EE フレームワークとの違い
- まとめ

はじめに： Web アプリケーション構築が抱える難題

Web アプリケーションの構築： J2EE 以前

J2EE™、特にその Web 層コンポーネント (サーブレットおよび JSP) は、第一世代の Web 開発者が抱える主な問題に対処して成功を収めています。J2EE 以前の世界では、第一世代の開発者は、単純なアプリケーションを構築するのでさえ、非常に多様なプログラミングモデル、API、サーバー固有の特徴と格闘しなければなりませんでした。企業規模のアプリケーションになれば、それが求める要件に対応する技術を選定するだけでも非常に多くの要素を検討する必要があったため、その難しさはさらに増します。実際にアプリケーションを構築することは、プラットフォームの未熟さ、API の不一致、クロスプラットフォームの統合の問題、優れたスケーラビリティを持つ開発・保守モデルの欠如によってさらに複雑な問題になっていました。

サーバーのベンダーは高レベルの強力アプリケーションフレームワークを提供することによってプログラミングおよび開発のスケーラビリティの問題の多くを解決しましたが、それらフレームワークには共有する基本前提がわずかしかなく、共通の規約やインフラストラクチャはありませんでした。一般にそれらのフレームはサーバーベンダーが専有するサーバーインフラストラクチャに束縛されていて、高機能で堅牢なエンタープライズアプリケーションの構築は可能にしたものの、ベンダーを変えたり、他のベンダーが提供する技術を簡単に利用したりすることはできませんでした。基本的に、アプリケーションを書き直すことなく、別のベンダーのプラットフォームに移行することは不可能だったのです。

企業の設計者はいくつかの戦略を採用してベンダーの囲い込みを回避しました。その中でも最も広まったのは、アーキテクチャーに大量の抽象化を組み込むという戦略です。この戦略によって問題の一部は解決され、エンタープライズビジネスオブジェクトおよびプロセスを Web コンテナの細部から切り離すことができましたが、他方で別の問題が生み出されました。抽象化によってアプリケーションが、場合によっては大幅に複雑さを増し、開発および配備の両面でマイナス面が生まれたのです。設計者が特定ベンダーが専有する API から距離を置こうとすればするほど、こうした複雑なインフラストラクチャをデバッグすることがますます難しくなっていきました。新しいプロジェクトのたびに、以前に出会ったものと互換性のない新しいアーキテクチャや制約条件が生まれるため、Web 開発者の技術で再利用可能なものはどんどん少なくなっていきました。

それぞれのアプリケーションはそれぞれの世界に引きこもり、特に土台のアプリケーションフレームワークが開発者にも設計者にも同様に強力な方向を提供していない場合、ほとんど一貫性がありませんでした。制約条件にはアプリケーション開発業務に集中するのに役立つものもありましたが、一部フレームワークが高度になり、開発者が高度な仕事だけでなく、時には日常的な仕事をこなす機能に取り組まざるをえなくなりました。

Web アプリケーションの構築： J2EE 後

J2EE の登場は、第一世代の Web アプリケーション開発に特有の問題の多くを解決しました。開発者は初めてコンテナと自身が開発したアプリケーションコンポーネント間の標準的な規約に頼ることができ、J2EE 準拠のあらゆるコンテナが綿密に設計された同一の API を提供する保証が得られたのです。設計者および開発者は、専有フレームや API、コンテナの無秩序な混在から自由になりました。

しかしながら、自由とともにあるのが責任です。J2EE はアプリケーションフレームワークの強固な礎ですが、J2EE 自体が、アプリケーションフレームワークであるわけではありません。J2EE の仕様では、アプリケーション開発の世界で推奨されていることを回避しています。アプリケーション開発ニーズに合うアプリケーションインフラストラクチャを設計 (または採用) するという重要な仕事を設計者と開発者に任せています。J2EE 単独では十分ではありません。

J2EE アプリケーションフレームワークの登場

実際のアプリケーション、特に大規模なエンタープライズ Web アプリケーションを開発する場合、必然的にある種のフレームワークを作成する必要があります。土台になる Web 固有のプラットフォームはすでに J2EE によって提供されていますが、実際のアプリケーション開発に必要なデルタを得る必要があります。社内でフレームワークを開発することは時間がかかるばかりでなく、誤りが多くなりがちです。

こうして、それほど再利用可能でないものも含めて、ある範囲の開発ニーズに対処しようとする J2EE 上で構築された再利用可能な Web アプリケーションが豊富に登場しました。たとえばあるフレームワークではもっぱらクライアントへのデータの生成に、また別のフレームワークでは入力データの検査に重点が置かれました。依然として、「fat クライアント」用と「thin クライアント」用の GUI 開発を統一しようとするフレームワークもありました。

第一世代の Web アプリケーション開発で使われていた事実上のアーキテクチャは J2EE によって廃止されたため、現在では、要件に最適のアプリケーションアーキテクチャを J2EE プロジェクトごとに評価し、選定する必要があります。たとえば タイプ I およびタイプ II サーブレットアーキテクチャ、Service to Workers 委託、MVC ベースの UI など、共通の概念および用語が育まれて、それらアーキテクチャを論議するのに役立つようになっています。しかしながら、こうした概念は基本かつ重要ではありますが、非常におおざっぱであり、非常に小さいものから非常に大きいものまで全範囲のアプリケーションに当てはまります。また、それら概念のどれも、繰り返し可能、保守可能、スケーラブルな方法での Web アプリケーション構築方法の細部を本当の意味では扱っていません。

最後の点を強調するなら、現代のほぼあらゆるアプリケーションフレームワークでは、実装、拡張性、開発者にかかる制約条件の面でそれらフレームワークの間に驚くほどの違いがあるのですが、タイプ II、Service-to-Workers、MVC ベースのアーキテクチャを使用するように主張されています。土台のアーキテクチャに関する知識は、開発者にフレームワークを紹介するのに役立つだけです。特にそれらフレームワークが対象とするアプリケーション規模が異なる場合は、開発者がフレームワークを学ぶのに役立つという点、さらには他のフレームワークとの比較という点でさえ驚くほど小さな一過性の役割しかありません。

現代のフレームワークは、それらの機能を実際に表現するのに十分な用語がある世界の向こう側にあります。つまり、フレームワークを比較したときにあるフレームワークが持つ特徴、特にエンタープライズアプリケーション開発でフレームワークが持つ機能を本当の意味で理解するには、はるかに細部にわたる分析が必要です。単に機能のチェックリストから始めるのは不十分です。

エンタープライズアプリケーションフレームワークの基準

Web アプリケーションの構築では、テスト済みで実証されたアーキテクチャに頼ることが非常に大切です。その大切さは、他のアプリケーション開発分野と比較すると、もっと増します。たとえば「fat クライアント」アプリケーションは、部分的に最適化されたアーキテクチャあるいは技術的な選択をしても、応答面でかなり許容されます。最も抽象化の度合いが高いクライアント側アプリケーションアーキテクチャでさえ、現代のワークステーションであれば十分なパフォーマンスで機能します。

しかし、同じアプリケーションが疎結合されたネットワークを介し、数百、数千人の同時ユーザーをサポートしている共有ハードウェアの上で動作する必要がある場合、必ずしもこのことは当てはまりません。こうした分野では、アーキテクチャあるいは技術の選択を誤ると、タイムリーに要求に応答するアプリケーションと、まったく応答しないアプリケーションとの間、あるいは開発と本稼働両面でうまくバランスがとれるアプリケーションといずれか一方でしかバランスがとれないアプリケーションとの間で決定的な違いが生まれます。

こうした要素の一部に欠陥があることは、少数の開発者が 1 つのアプリケーションで密に連携する小中規模のアプリケーション開発では許されることもありえます。そうした場合は、将来の変更対象範囲や他のチームとの連絡が限られたものであり、チームのメンバーが簡単に調整して問題が起きたときに対処できるため、アーキテクチャーはずっと流動的になる可能性があります。コーディング基準および最良の手法がその場で簡単に共有され、アプリケーションの古い部分の改造が数時間の仕事ですむためです。チームの技術レベルはほぼ同じで、ずっと安定していて、長期にわたって一緒に活動する傾向があります。そうしたアプリケーションのユーザーベースは少人数であったり、応答時間が鈍くても許容されるため、通常パフォーマンスは重要な要素ではありません。

エンタープライズアプリケーションの開発は、小中規模アプリケーション開発モデルの正反対です。チームの連絡には手間がかかり、アプリケーションの変更は大規模な波及効果を生みます。チーム全体に最良の手法が広まることはなく、アプリケーションの部分的な改造は単純に不可能になります。開発者の回転率は高く、開発者のもつ技術力はかなり多様です。またパフォーマンスについては、ユーザーがアプリケーションを使い続けるか、あるいは逆に便利さは劣るがもっと応答性に優れた代わりのものを求めてアプリケーションを替えるかを決定する役割を果たす可能性があるため、非常に重要です。

エンタープライズ Web アプリケーション開発を容易にするフレームワークはどれも、その設計および実装の両面でこれらの制約条件を考慮する必要があります。さらに重要なことに、それら制約条件がアプリケーション開発業務に与える影響を最小限に抑えるか、理想的にはなくす必要があります。

このため、どのようなエンタープライズフレームワークであっても、以下のことを行う必要があります。

- アプリケーションに一貫性をもたせ、チームやプロジェクト、会社にまたがって開発者の技術を再利用できるようにする。アプリケーション開発を始める際の明確な取り組み姿勢が必要ですが、これが開発者の能力を制限するものであってはならない。
- 高低両レベルの機能を用意して、チームがそれぞれの要件に合わせて適切なバランス点を見つけ、時間的制約の中でプロジェクトを完成できるようにする。
- アプリケーションの保守性を向上させる具体的な手段を用意し、設計者と開発者がその仕事を自分でする必要がないようにする (できれば、異なる手段をいくつも用意する)。
- 必ずしも開発者の誰もが同じ知識と経験を持っているわけではないので、未経験の開発者の指導をする。Web アプリケーションの分野、さらには企業開発の分野は初めてという開発者がいるかもしれない。
- 上級開発者の補助をして、フレームワーク機能に触れずに、土台の J2EE プラットフォームを直接利用して高度な機能を作成できるようにする。
- 企業の設計者に要請して、アーキテクチャ面で組み込む必要がある高度な要素をフレームワークですぐに利用できるようにしたり、フレームワークと互換性を持たせるようにする。アーキテクチャ面の要素の選択が慎重に行われること、できればリソースを前提に設計者と同じ選択が行われることが望ましい。

最も大切なことは、フレームワークが企業環境の中で実証済みで成熟しており、堅牢でパフォーマンスに優れていなければならないということです。フレームワークのユーザーは想定されていることを認識し、開発の仕事を始める前にフレームワークがそれらの条件を満たしていることを確認済みである必要があります。

この章の以降の部分では、真のエンタープライズクラスの Web アプリケーションフレームワークであることを目標に掲げる Web アプリケーションフレームワークが、どのようにこうした問題に取り組み、求められる基準を満たしているのかについて説明します。

Web アプリケーションフレームワークとは

概要

Web アプリケーションフレームワークは、エンタープライズ Web アプリケーション開発用に作られた、標準に基づく強力で成熟した J2EE Web アプリケーションフレームワークです。Web アプリケーションフレームワークでは、表示フィールドやアプ

リケーションイベント、コンポーネント階層、ページ主体の開発手法などの既存の概念と、Model-View-Controller および Service-to-Workers パターンに基づく最新技術を結集した設計手法とが一体化されています。

Web アプリケーションフレームワークは、業界をリードするソフトウェア技術者やコンサルタント、Web アプリケーション開発者、企業の Web 設計者の集団的な経験に基づいています。2000 年 1 月から開発に入り、2000 年 6 月から一般に利用できるようになりました。それ以来、Web アプリケーションフレームワークは、現実に数十の Web アプリケーションで使用され、毎日、数百万のユーザーに対応して、数百万ドルの金融取引が行われる実際のサイトで成功を収めています。

Web アプリケーションフレームワークが対象とする利用者

Web アプリケーションフレームワークは、主として、中大規模および巨大規模の Web アプリケーションを構築する J2EE 開発者のニーズに応えることを意図しています。Web アプリケーションフレームワークは小規模の Web アプリケーションにも使用でき、実際そのように利用されていますが、その主な利点は、小規模ではそれほどすぐに明らかになりません。Web アプリケーションフレームワークは、アプリケーションを長期間維持して、多数の変更を加え、その中で成長していくとき特にその機能を発揮します。簡単に言えば、Web アプリケーションフレームワークは、エンタープライズアプリケーションの開発支援の面で卓越しています。

Web アプリケーションフレームワークは、再利用可能なコンポーネントの中核となる機能を提供するため、簡単に Web アプリケーションに組み込むことが可能な、既成のコンポーネントを提供することを望む開発者に適しています。Web アプリケーションフレームワークが垂直型 Web 製品を構築するためのプラットフォームとして非常に適しているのは、こうした理由からです。特に、そうした拡張機能によって、ユーザーとオリジナルの開発者はともに、既存の垂直型機能を拡張したり、活用したりするための綿密に定義された手段を手にすることができます。

Web アプリケーションフレームワークができること

Web アプリケーションフレームワークは、最新技術を結集した J2EE 設計パターンを利用したエンタープライズ Web アプリケーションを構築する開発者を支援します。Web アプリケーションフレームワークは設計パターンに基づく骨組みを提供し、企業の設計者は自身のアーキテクチャの他の部分を組み込むことができます。Web アプリケーション開発者は簡単な開発手法を見出し、企業設計者は綿密に定義された方法で他のエンタープライズ層およびエンタープライズコンポーネントと統合する、明確に記述された設計手法を見出します。

Web アプリケーションフレームワークは、高低両レベルのインフラストラクチャと設計パターンばかりでなく、完全なコンポーネントモデルを提供することによって開発者が再利用可能なコンポーネントを構築する支援をします。開発者が定義したコンポーネントは、ネーティブコンポーネントであるかのように Web アプリケーションフレームワークと対話する最高級のオブジェクトです。コンポーネントは自由に組み合わせて、1 つのアプリケーション全体、異なるアプリケーション間、さらにはプロジェクトや企業にまたがって再利用できます。

まとめとして、Web アプリケーションフレームワークは、J2EE 開発初心者が Web アプリケーション開発に初めて取り組むのに役立ちます。また、上級の J2EE 開発者は、他のフレームワークで実現不可能な高度な機能を開発するための強力なツールキットを利用することによって、その能力を高めることができます。

Web アプリケーションフレームワークができないこと

Web アプリケーションフレームワークはエンタープライズ層のフレームワークではありません。このことは、EJB や Web サーバー、その他の種類のエンタープライズリソースの作成を直接支援するわけではないことを意味します。Web アプリケーションフレームワークはエンタープライズアプリケーション開発向けに作られていますが、正しくはエンタープライズ層リソースのクライアントであり、それらリソースを利用するための本格的な第一級の仕組みを提供します。

Web アプリケーションフレームワークの仕組み

設計パターンの利用

Web アプリケーションフレームワークは、最新技術を結集した、業界に広く受け入れられている設計パターンおよび設計手法に基づいてます。J2EE のプレゼンテーション層フレームワークとして、JavaSoft が発表したコア J2EE パターンを実装し、またそれらパターンに深く依拠しています。次の表は、Web アプリケーションフレームワークにおける、発表されている J2EE 設計パターンの利用状況をまとめています。

Intercepting Filter	プレゼンテーション	コア J2EE パターン (http://java.sun.com/blueprints/corej2eepatterns/index.html を参照)	Java BluePrints パターンカタログ (http://java.sun.com/blueprints/patterns/catalog.html を参照) *	Web アプリケーションフレームワーク実装
Intercepting Filter	プレゼンテーション	○	○	Servlet 2.3 フィルタ
Front Controller	プレゼンテーション	○	○	○
Composite View	プレゼンテーション	○	○	○
View Helper	プレゼンテーション	○	○	○
Dispatcher View	プレゼンテーション	○		○
Service To Worker	プレゼンテーション	○		○
Business Delegate	プレゼンテーションとビジネス	○	○	○
Session Facade	ビジネス	○	○	**
Service Locator	ビジネス	○	○	**
Value List Handler	ビジネス	○	○	**
Composite Entity	ビジネス	○	○	**
Transfer Object Assembler	ビジネス	○		**
Transfer Object	ビジネス	○	○	**
Service Activator	インテグレーション	○		**
Data Access Object	インテグレーション	○	○	○
Fast Lane Reader			○	○

Intercepting Filter	プレゼンテーション	コア J2EE パターン (http://java.sun.com/blueprints/corej2eepatterns/index.html を参照)	Java BluePrints パターンカタログ (http://java.sun.com/blueprints/patterns/catalog.html を参照) *	Web アプリケーションフレームワーク実装
Model-View-Controller	プレゼンテーション		○	○
Adapter	すべて			○
Command	プレゼンテーション			○

* J2EE BluePrints サンプルアプリケーション (PetStore など) には、ビジネスおよびインテグレーション層の多数のパターンが実装されていますが、事実上それらのパターンはデモの目的にのみ実装されています。また、それらパターンの多くは、EJB の使用向きです。J2EE BluePrints アプリケーションはこのことを前提にしていますが、Web アプリケーションフレームワークはそのことに囚われません。いくつかの理由から、Web アプリケーションフレームワークには、いくつかの特殊なケースで EJB の使用に代わる手段が用意されており、それらの代替手段では、Web 層エンティティからではありますが、ビジネスおよびインテグレーション層のパターンの一部を利用します。

** Web アプリケーションフレームワークにおける J2EE プレゼンテーション層のパターンの実装は最小限です。ただし、プレゼンテーション層専用のフレームワークから期待されるのとは異なり、このことは必ずしもビジネスおよびインテグレーション層のパターンが実装されていないということではありません。これらのパターンの大半は、推奨する最良の手法としてエンタープライズおよびインテグレーション層開発者に任せられており、フレームワークのプレゼンテーション層のパターンと完全互換で、それらのパターンと統合することができます。

上記のパターンの使用に加えて、Web アプリケーションフレームワークは N 層 / サーブレットアーキテクチャに基づいており、完全に、それらのパターンを反映したインタフェースおよびオブジェクト規約に基づいて設計されています。すなわち、Web アプリケーションフレームワークは、第 1 に協力し合う設計パターンを統合したフレームワークであり、第 2 にそれらパターンを実装したフレームワークです。

Web アプリケーションフレームワークのパターンで第一のパターンは、Model-View-Controller (MVC) パターンです。他のフレームワークでは、MVC フレームワークであることが主張されていますが、実際には、それらフレームワークは、通常、パターンのコンポーネントのうちの 1 つないし 2 つに重点を置いているだけで、3 つすべてに重点を置いていることはほとんどありません。また、他のフレームワークでは、たいていカスタムライブラリとともに JSP が完全で適切なビュー層で構成されていると主張している場合がありますが、その一方で、アプリケーション専用のビジネスオブジェクトは、完全で適切なモデル層で構成されているとも主張している場合があります。

これらはもっとらしい主張ですが、実際は違います。

Web アプリケーションフレームワークは、MVC パターンの 3 つのコンポーネントすべてに完全に対応しています。具体的な関係で正式なビューおよびモデルエンティティを定義し、アプリケーションが適切な方法でコントローラロジックのスコープを決めることを可能にする高度な論理コントローラロールを提供します。Web アプリケーションフレームワークのビュー層には JSP 技術が反映されていますが、JSP 技術と同じではありません。同様に、モデル層には他の J2EE 技術が反映されていますが、そのどの技術とも同じではありません。これらの、そして以下に説明する理由から、Web アプリケーションフレームワークは、単純に他のフレームワークでは実現できない、前例のない拡張性を開発者に提供します。

機能の種類

論理的には、Web アプリケーションフレームワークの機能に分類されます。

- Web アプリケーションフレームワークコア
- Web アプリケーションフレームワークコンポーネント
- Web アプリケーションフレームワーク拡張

Web アプリケーションフレームワークコア

通常、Web アプリケーションフレームワークコアは、単に「Web アプリケーションフレームワーク」とだけ呼ばれます。コアは、基本インタフェース、オブジェクト規約、プリミティブばかりでなく、Web アプリケーションフレームワークアプリケーションに最低限必要なインフラストラクチャを定義しています。コアがコンポーネントライブラリを提供することはありませんが、コンポーネント作成者にその対応技術を提供します。Web アプリケーションフレームワークコアに含まれているのは、コンテナビューやタイルビュー、ツリービューなどのビューベースのプリミティブの他、データセットモデルや照会モデル、ツリーモデルなどのモデルベースのプリミティブです。Web アプリケーションフレームワークコアはまた、要求送信や再利用可能なコマンドオブジェクト用のプリミティブも提供します。これらのプリミティブを利用し、開発者は同じプロジェクト内あるいはプロジェクトにまたがって共有することが可能なアプリケーション専用のコンポーネントや再利用可能なコンポーネントを簡単に作成することができます。Web アプリケーションフレームワークコアにはまた、開発者がすぐに高機能のアプリケーションの構築を始めることを可能にする高レベルの機能も含まれています。以下の項では、これらの機能をさらに詳しく取り上げます。

Web アプリケーションフレームワークコンポーネント

Web アプリケーションフレームワークコンポーネントは Web アプリケーションフレームワークコアのインフラストラクチャを活用して、アプリケーション開発用の高レベルの再利用可能なコンポーネントを提供します。それらのコンポーネントには、用途範囲の違いを考慮してさまざまな風合いのものがあります。たとえば水平型の

Web アプリケーションフレームワークコンポーネントは、全体としてコンポーネントの中でも特に汎用のコンポーネントであり、その利点は柔軟性とカスタマイズの容易さにあります。この種のコンポーネントは、多くの Web アプリケーションフレームワークユーザーがプロジェクトや企業にまたがって利用することができ、一般に特定のルック & フィールに偏っていません。垂直型の Web アプリケーションフレームワークコンポーネントは特定の用途向けに作られており、高レベルの機能と使い易さを提供することを可能にします。この種のコンポーネントの利用可能性は限定されますが、その用途範囲が綿密に定義されているため、パラメータ部分が最低限ですみ、特定のルック & フィールを採用することができます。Web アプリケーションフレームワークコンポーネントはどれも、Web アプリケーションフレームワークコアが提供し、WebAction や SQL ベースのモデル実装、ツリービューなどのその高レベルの機能に基づく機能のすべてを利用することができます。

Web アプリケーションフレームワーク拡張

Web アプリケーションフレームワーク拡張を使用することによって、Web アプリケーションフレームワークとの互換性を維持しながら、非 J2EE 機能を利用することができます。多くの場合、Web アプリケーションフレームワーク拡張は、Web アプリケーションフレームワークアプリケーションからシームレスにコンテナ固有の機能を利用することを可能にします。拡張は、アプリケーション開発ではなく、技術の統合に重点を置いているという点で、Web アプリケーションフレームワークコンポーネントと異なります。

技術概要

Web アプリケーションフレームワーク、もっと正しくは Web アプリケーションフレームワークコアは Pure Java であり、業界標準の JAR ファイルでパッケージ化されています。

Web アプリケーションフレームワークでは、最上位のパッケージを以下のように定義しています。

- `com.iplanet.jato` - 要求処理インフラストラクチャ
- `com.iplanet.jato.command` - コマンド関連のインタフェースと実装
- `com.iplanet.jato.taglib` - カスタム JSP タグライブラリ
- `com.iplanet.jato.model` - モデル関連のインタフェースと実装
- `com.iplanet.jato.view` - 一般的なビュー関連のインタフェースと実装

これらのパッケージにはそれぞれに、HTML 専用のビュー実装や SQL 専用のモデル実装などの特定用途向けの派生物からなるサブパッケージが含まれています。Web アプリケーションフレームワークコンポーネントや Web アプリケーションフレームワーク拡張には正式なパッケージあるいはクラスはありません。これは純粋に論理的な分類です。

Web アプリケーションフレームワークアプリケーションの作成では、開発者は既存の Web アプリケーションフレームワーククラスからアプリケーションに固有のサブクラスを取り出すか、アプリケーションに固有の方法でいくつかの Web アプリケーションフレームワークインタフェースを実装します。たいていの場合は、既存の Web アプリケーションフレームワークコア実装をスーパークラスとして使用し、大量の有用な動作を継承します。(コンポーネント開発者の場合は、一群の Web アプリケーションフレームワークインタフェースを直接実装することがよくあります。)

アプリケーションオブジェクトは、ページという概念を軸に編成します。各ページは、生成指定 1 つ (通常は静的コンテンツとマークアップ、カスタム Web アプリケーションフレームワークタグからなる) と、ページのビュー階層のルートからなるクラス 1 つから構成されます。サーバーに要求を行うと、結果として 1 つのページが返されます。アプリケーション経由のページの流れは、開発者が作成した制御ロジックによって決まります。ページ同士の間に、開発者が規定する以外の固定した関係はありません。

HTML の世界では、一般に、生成された各ページには、ユーザーが起動することができるリンクまたはボタンが 1 つ以上含まれます。リンクまたはボタンを起動すると、サーバーにデータが送信され、その起動に固有のコマンドオブジェクトが起動されます。このコマンドオブジェクトはそれ自体が処理を行うか、開発者の定義したイベントメソッドに要求の処理を委託することができます。最終的には、要求は、クライアント向けに応答を生成する仕事をするリソースに転送されます。

たいていの場合、このリソースは、動的コンテンツの生成に Web アプリケーションフレームワークタグライブラリを使用した HTML 形式の JSP ページです。タグライブラリは、Web アプリケーションフレームワークのビューコンポーネントを使用して、それが生成するデータを取得します。これらのビューオブジェクトは、1 つ以上のモデルオブジェクトに関連付けられ、必要に応じてそれらのモデルからデータを引き出します。このように Web アプリケーションフレームワークのビューは、任意の個数のモデルの階層的なファサードとして機能し、複数のページにまたがり、異なるモデルとの組み合わせで再利用することができます。一般にモデルは、表示やビューの依存関係がないため、任意の個数のビューで利用することができます。

ページの形式で応答を受け取ると、ユーザーがリンクまたはボタンを起動し、その結果、要求が Web アプリケーションフレームワークアプリケーションに送信されます。このとき要求の送信先は、ページを生成したのと同じオブジェクトです。このため、Web アプリケーションフレームワークインフラストラクチャは、送信データを、そのデータが元々あったのと同じビュー (およびモデル) にマップすることができ、こうして、そのデータの仮想持続性を提供します。開発者は、応答から要求のサイクルが間に存在しなかったかのようにアプリケーションオブジェクトおよび送信データと対話します。データがもとのオブジェクトにマップし直されると、そのリンクまたはボタンの起動に固有のコマンドオブジェクトが起動され、再びサイクルが開始されます。

Web アプリケーションフレームワークと他の J2EE フレームワークとの違い

以下の節では、Web アプリケーションフレームワークと現代の他の Web アプリケーションフレームワークとの主な違いを簡単に説明します。

J2EE 規格準拠

多くのフレームワークでは、実用的な技術としてサーブレットを採用しながら、JSP を避けたり、その逆に JSP を採用しながら、サーブレットを避けたりしています。その他、これらの規格を採用しているというフレームワークもありますが、現実には、お粗末なサーブレットレベルの統合で J2EE コンテナ内で実行できる一社専有のコンテナにすぎません。

Web アプリケーションフレームワークは、サーブレットや JSP などの J2EE 規格を直接取り入れる一方で、開発者が自由に J2EE が提供する機能を利用することを可能にしています。Web アプリケーションフレームワークはコンテナ内のコンテナではなく、J2EE から開発者の注意をそらすことを意図した 1 つの層でもありません。Web アプリケーションフレームワークは企業の Web アプリケーション開発を容易にする J2EE 機能を補強しながら、開発者が必要に応じて J2EE/Web アプリケーションフレームワークと自由に対話できるようにしています。

既存のパラダイム

Web アプリケーションフレームワークは、表示フィールドやアプリケーションイベント、コンポーネント階層、ページ主体の開発手法を提供しており、それらのすべてが時間の試練を経ていて、Swing や Delphi、Visual Basic、あるいは PowerBuilder を使ったクライアント側アプリケーション開発に精通している開発者には非常に快適に扱えます。Web パラダイムによる違いはありますが、それらの既存の構成によって、開発者には Web アプリケーションフレームワークは違和感なく扱えるものになっており、アプリケーション開発の実質的なスピードアップになります。こうしたことはまた、Web アプリケーションフレームワークが特に Forte for Java や JBuilder などのアプリケーションビルダーとの統合にも適していることを意味します (アプリケーションビルダーとの統合の話題については、「ツール対応」の項を参照)。

アプリケーションの一貫性

現代の多くの Web アプリケーションフレームワークはきわめて柔軟性に富み、場合によっては、そのことが設計の根本的な意図になっています。モデル層などの、アプリケーションのいくつかの面について意識的に非規範的であるように努めています。アプリケーション設計のいくつかの分野 (特に一般的なところでは MVC アーキテクチャのコントローラとビューの部分) に重点を置き、残りは開発者に任せています。

設計者や開発者の中には、柔軟性は欠点ではないと主張する人もいるかもしれません。しかし、企業開発のことを考えると、そのことが確実に欠点になる可能性があります。当初は、柔軟性を欠点とすることは奇妙に聞こえるかもしれませんが、柔軟性とアプリケーションの一貫性の間には逆の関係があります。J2EE API 自体がそうですが、最大の柔軟性を持つフレームワークは、開発方法が広範囲に異なるアプリケーションになることを意味します。

柔軟性に制限がなかったり、開発の方向が綿密に定義されていなかったりすると、未熟な設計者や開発者が、手近に仕事をこなせるように見える手法 (最終的にはその手法に欠陥があっても) を発見することがあります。開発作業の全範囲を通じて従うべき明確な道筋をフレームワークが少なくとも 1 つでも提供できないと、開発者はおそらくフレームワークが提供する利点を活かすことができない手法を採用することになるでしょう。また、孤立したチームがそれぞれに異なる手法を見つけ、このために同じアプリケーション内でも、簡単にあるグループが他のグループの作業を理解したり、維持したりできなくなります。最悪の場合は、アプリケーションの一部分における手法の欠陥が他の部分に影響し、アプリケーション全体がパフォーマンスやスケーラビリティの問題を抱えるまでになります。アプリケーションの全体的な方向として、未熟な設計者や開発責任者が正しくない方向を選ぶと、こうした状況は簡単に起こります。この選択はアプリケーション全体を通じて、扱いにくいアーキテクチャ上の問題を引き起こし、最悪の場合には、アプリケーションが最終的に機能しなくなるという結果をもたらすこともあります。

Web アプリケーションの開発者にとっては残念なことに、大部分のフレームワークは、開発および本番の両面で簡単に逆効果になる可能性がある方法で柔軟であることが判明しています。上記のように、優れたエンタープライズフレームワークは、未熟な開発者が上級の開発者の邪魔をすることなしに積極的な方向に向かうように導くべきです。多数のフレームワークは後者は達成していますが、それは、設計思想あるいは設計の欠陥のいずれかの原因で、アーキテクチャあるいはアプリケーション開発のいくつかの面でそれらフレームが非規範的であるためです。このため、開発者は、プロジェクトを開始する際に多くの選択をしなければならず、そこには、正しい選択をしないと、プロジェクト全体をかなり危険にすることが含まれています。

これに対し Web アプリケーションフレームワークは、他の手法を排除することなく、Web アプリケーションアーキテクチャとアプリケーション開発の両方で実証済みの暗黙の方向を提供します。このことは、アプリケーションとの対話点を綿密に定義しているばかりでなく、既存の動作を拡張、増強、置き換える方法を明確に定義することによって実現されています。Web アプリケーションの開発に Web アプリケーションフレームワークと 別のフレームワークを利用することの決定的な違いは、Web アプリケーションフレームワークでは、初めての利用者が、間違う可能性もある選択を行うことがないということです。初心者は最初から一般的な手法を目にする

ことができ、進歩して、Web アプリケーションフレームワークおよび J2EE に慣れてくると、他の手法や技術が見えるようになります。また、開発者がそうなるまでに行った作業はどれも、後で使用する高度な技術と一貫性があります。この結果、Web アプリケーションフレームワークで作成したアプリケーションは、他のフレームワークで作成したアプリケーションよりも相互にずっと似通っています。高および低レベルの機能の使用の両面で矛盾が少なく、このため保守が容易です。

表示 / 送信処理のバランス

現代の多くのアプリケーションフレームワークは、一般に非常によく受け入れられている技術であるカスタムタグライブラリから発展してきたものです。場合によってはカスタムタグライブラリと 1 つないし 2 つの追加インタフェース以上のものがほとんどないフレームワークもあります。このため、そうしたフレームワークは、ユーザー向けのデータの表示にひどく偏り、ユーザーからのデータの処理の支援がほとんどないという点で近視眼的です。

そうしたフレームワークはおそらく一定の開発者ニーズにはよく応えているのでしょうが、他のニーズは犠牲にしていることでしょう。タイプ II アーキテクチャの場合、JSP などの生成技術は送信 (要求) サイクル中に全く関与しません。このことは、JSP でしかビュー表現が定義されていない場合、送信サイクルのロジックがそのビューを利用できないことを意味します。その場合、このロジックは、入力として未加工のパラメータリストを受け取るだけです。開発者は、ほとんどあるいはまったく手助けなしに、未加工の形式でそれらの値を使用することになります。突然に深いところから最も基本的なサーブレット技法に入り込むことになるのです。

こうしたフレームワークで頻繁に見られるのは、アプリケーションオブジェクト間の明確な関係の指定がなく、維持することが難しいことです。受信データに対する構造がほとんどまたはまったく提供されていないため、起動されたコンポーネントが暗闇で仕事をせざるをえず、与えられた要求の起動でどのようなデータを受け取るのか確実に知ることができません。このことは、プロジェクトの開始時点では、すぐに負担になることは明らかにならないかもしれませんが、アプリケーションが成長するにつれて短期間に大ききな問題になります。

表示サイクルと送信サイクルのバランスが欠けているために、オブジェクト間の依存関係が激増するというのはよくあることです。一般に、この関係の増大は、ターゲットオブジェクトまたはバックエンドに、手動で入力データをシャッフルする以上のことをしない必要がある低レベルのコントローラロジックの増殖に反映されます。そして、このことは、バックエンドオブジェクトまたはモデルを使ってページを生成するという不均衡な考え方につながる可能性があります。しかも、そのオブジェクトまたはモデルは、以前に生成したページからの要求を処理する際に直接使用されるわけではありません。この不均衡により、開発者にはさらなる負担がかかり、バックエンドコンポーネントを細かく管理し、Web アプリケーションコンテナの実行という低レベルの細部を心配しなければならなくなります。

表示主体のアーキテクチャの犠牲になるのは、生産性と保守です。これに対し、Web アプリケーションフレームワークは、その正式なビュー層により、バランスのとれた方法で表示サイクルと送信サイクル両方の支援をします。他のフレームワークでは、ビュー層が JSP または他の種類のコンテンツ生成技術として大まかに定義されているのに対し、Web アプリケーションフレームワークでは、生成指定 (JSP) とビューコンポーネントを明確に区別しています。これらが一体になったものだけが完全なビュー層と見なされます。Web アプリケーションフレームワークアプリケーションでは、ビューコンポーネントからなる階層を第一に定義し、その後、生成指定からそれらコンポーネントを参照します。開発者は、表示および送信両方のサイクルにおいて同じ方法でビューコンポーネントと対話します。ビューコンポーネントは標準的なビューフォームです。

正式なモデルエンティティ

前項で説明したように、多くのフレームワークはユーザーへのデータの表示を支援する技術に重点を置いています。この種のフレームワークでよくあるのは、XML と XPath に重点を置いたものです。最新の技術に期待を寄せる開発者には魅力的ですが、これらのフレームワークは、アプリケーションの送信サイクルで開発者に提供するものはほとんどなく、しばしば、XML または他の表示指向の形式でアプリケーションデータを表現することを要求します。アプリケーションデータをフレームワーク主体の表現に強制的に変換することは困難な作業であり、場合によって致命的な欠点になります。

これに対し Web アプリケーションフレームワークでは、アプリケーションはそのデータをビューに囚われない方法で表現し、特定のデータ形式を意味することなくそのデータを取得する正式な仕組みを提供すべき、という考え方を取っています。Web アプリケーションフレームワークは、あらゆるメソッドが実装する必要がある少数の標準のメソッドを定義した正式なモデルエンティティを提供しているのはこのためです。任意のモデル固有のキーを使用し、モデル使用者 (Web アプリケーションフレームワークのビューなど) はそのモデル内部でのモデルデータの表現方法に関係なく、標準的な方法でモデルデータを取得することができます。

このため、Web アプリケーションフレームワークコンポーネントは同じ方法で任意のモデルと対話でき、同じビューに別のモデルをプラグインすることを可能にします。モデルは互いに交替可能になり、このため、それらモデルが表すデータも交替可能になります。純粋に表示の目的に特定の形式にデータを操作する必要はなく、ビュー層が、その対話相手のデータの特定の型を理解する必要もなくなります。ビュー層がモデルのネーティブのデータ形式間の違いを認識することはなく、1 つのアプリケーション内にさまざまな種類のモデルが共存できます。モデル使用者からは XML/XPath や JDBC、JDO、その他エンタープライズデータのすべての同じに見え、このため、Web アプリケーションフレームワークは、フレームワークがこれらデータ形式の 1 つに重点を置いていても、そのフレームワークの開発手法を組み込むことができます。

まとめとして、エンタープライズ層のリソースにモデル構造を介在させることことによって、アプリケーション設計の一貫性をはるかに高めるばかりでなく、保守の負担を大幅に軽減する度合いの抽象化が適用されます。エンタープライズ層から使用可能なデータの正式な定義に際し、開発者はまた、アプリケーションの層間の正式ではあるがまだ疎結合の規約を定義します。この規約により、将来のアプリケーションの改造は簡単になり、しかも綿密に定義された方法で改造することが可能になります。後退が発生する率は低くなり、発生しても、すぐに明らかになります。

アプリケーションイベント

Web アプリケーションフレームワークは、アプリケーション関係の発生事象に対する多数のイベントを提供します。それらイベントは、一般要求イベント、特定要求イベント、表示イベントの 3 つの種類があります。

一般要求イベントは、`onBeforeRequest()` や `onSessionTimeout()`、`onUncaughtException()` などのイベントです。開発者は、これらのイベントを使い、一般的なアプリケーションおよび要求ライフサイクル事象やエラー状態に応答することができます。デフォルトではエラー関係のイベントは、ローカライズされた統一性のある仕組みを使ってユーザーにエラーを報告します。開発者は、この種のイベントがアプリケーションに固有のアクションを取るように置き換えることができます。

特定要求イベントは、ユーザーアクションに基づいて発生します。ユーザーがページ上のリンクまたはボタン (Web アプリケーションフレームワークではコマンドフィールドともいう) を起動すると、その要求によって、起動に対するコマンドオブジェクトがサーバー側で起動されます。そうしたアクションに対する応答として、ユーザーは独自のコマンドオブジェクトを提供できますが、デフォルトのコマンド実装では、要求の処理は `handle<`*name*`>Request()` の形式の要求処理イベントメソッドに委託されます (<*name*> はユーザーが起動したコマンドフィールドの名前)。このイベントは、コマンドフィールドの親コンテナ上で起こされ、こうして、元々リンクまたはボタンを生成したコンポーネントのスコープに限定されます。このイベントハンドラ内では、開発者は任意のアクションをとることができ、要求を直接処理するか、別のオブジェクトに要求の処理を委託するかします。

この Web アプリケーションフレームワーク機能と類似の要求処理機能の最大の違いは、イベントが属するコンポーネント上でイベントが起こされ、要求処理機能がリンクまたはボタン単位にきめ細かくなっていることです。一般に他のフレームワークは HTML フォームごとにきめの粗いイベントハンドラを 1 つだけ提供し、ユーザーのアクションに基づく条件付きのコードを作成することは開発者の仕事になります。このため、一群のフィールドの変更でのコードの保守は煩雑で困難です。また、この手法は、モジュール式で自己完備型のコンポーネントの利用を困難にします。なぜなら、その 1 つのイベントハンドラがコンポーネントに含まれているかどうかに関係なく、フォームに対してリンクまたはボタンの追加あるいは削除が行われるたびに、イベントハンドラを変更しなければならないためです (この欠点の詳細は、「階層ビューとコンポーネントのスコープの限定」の項を参照)。

Web アプリケーションフレームワークはきめの細かいフィールドレベルの表示イベントを提供します。表示イベントはページの生成中に起こされ、単純に他の方法で不可能な生成プロセスへのフックを開発者に提供します。そのため、開発者は表示イベントからタグハンドラ、さらには JSP のページコンテキスト、出力ストリームにアクセスすることができます。表示イベントを使用して、フィールドの生成を飛ばしたり、現在の生成ページを完全に打ち切ったりできます。また、高度なコンテンツ選別機能を用意すれば、JSP が生成する発信コンテンツを加工したりできます。また、表示イベントは、コンポーネント内部からそのコンポーネントに関係する表示ロジックをカプセル化し、そこで高度な生成技術が使用されていても、高い度合いでコンポーネントを再利用可能にすることを可能にします。

さらに重要なことに、表示イベントは JSP から Java コードとプログラムのような構造を排除します。一般に、JSP 内のあらゆる種類のプログラムコンストラクトは、保守に問題があります。これは、JSP 作成者にアプリケーション機能が明かされることと、潜在的に多数の場所で並列コンテンツによってこの機能の複製を作成する必要があるためです (並列コンテンツの詳細は、「並列コンテンツのサポート」の項を参照)。この種の機能は、大きな保守を必要としないか、限られた寿命の小中規模のアプリケーションには、生産性の利点があると考えられますが、企業では、そうしたアプリケーションは一般的ではありません。多くのフレームワークはこの種のアプリケーション開発を強調し、それらフレームワークの機能の多くは従来のプログラミング言語に似たプログラム構成一式でそれらの機能を達成することを目指しています。これに対し Web アプリケーションフレームワークでは、JSP の長所を認識し、保守性あるいはアプリケーション開発者が JSP の生成を細かく制御する能力を傷つけることなく、それらの長所を活用します。

階層ビューとコンポーネントのスコープの限定

大部分のフレームワークでは、HTML フォーム内でデータフィールド名用に平面の名前空間を使用します。この平面の名前空間によって、ビューコンポーネントの組み合わせ方法は大きな制約を受けます。たとえば同じフォーム上で「name」という名前のフィールドを使用するコンポーネントを 2 つ使用することはできません。これに対する最後の手段としては、あらゆるコンポーネントおよびフォームにまたがってグローバルに、すべてのフィールドに対して一意の名前を考え出すという方法が残されているだけです。この回避策が開発の途中でうまくいかなくなることは明らかです。グローバルなコンポーネント市場の発展を阻害します。

密結合されたフォームとオブジェクトの調和に頼っているフレームワークの場合、この状況はさらに悪くなります。この状況では、各 HTML フォームは、フォームのフィールドごとに補助メソッドとミュテータメソッドを持つ 1 つの Java オブジェクト (通常は JavaBean) に対応します。「name」などの単純なフォームフィールド名は、`getName()` や `setName()` などの Java メソッドに簡単にマップされます。しかし、上記したように、再利用可能なコンポーネントを使用しようとする場合、開発者はそうした単純な名前をめったに利用できず、グローバルに一意の名前を利用する必要があります。`com.foo.componentA.name` など、一意にするために作った複雑なフィールド名の、Java メソッド名へのマッピングは、整然としているとは言えませ

ん。そうした名前は Java メソッド命名基準に準拠する必要があり、実用的な選択肢として `getCom_foo_componentA_name()` や `setCom_foo_componentA_name()` があるぐらいです。利用することはできますが、整然性や保守性は劣ります。

フォームのフィールド名に対するファサードとして単一オブジェクトに頼るフレームワークはどれも、ビューコンポーネントの利用を完全に排除しています。すなわち、フォームの一部が複数のページで使われているかどうかに関係なく、ページまたはフォームで使用されているあらゆるデータを、単一のオブジェクトインタフェースで表す必要があります。開発者は 1 つのフォームでの使用のためにだけ 1 つのオブジェクトを作成して、そのオブジェクトを他のもっと再利用可能なオブジェクトに委託することになるかもしれません。しかし、このためには退屈で保守しにくいデータシャッフルコードが必要とされ、真のコンポーネントアーキテクチャではありません。また、フォームまたはページで変更を加えるには、コンパイル作業が必要になり、アプリケーションビルダーとの連携を求めるフレームワークの重大な短所になります。

これに対し Web アプリケーションフレームワークは、密結合されたフォームとオブジェクトの調和に依拠しない階層名前空間を HTML フォームのフィールドに提供します。各表示フィールドビューは、親コンテナビューの子としてそれぞれに別に作成され、コンテナ内で単純なローカル名を使用します。つまり、限定された一意のグローバル名を暗黙で継承します。これらの限定されたフィールド名は、他のコンテナ内でローカル名が同じであったとしても、他のフィールド名と衝突しないことが保証されています。このため、独立したビューコンポーネントを自由に組み合わせることができ、互いに衝突することはありません。Web アプリケーションフレームワークでは、送信サイクル中に自動的に、それら限定フィールド名に関連付けられているフォームデータの、コンポーネントへのマッピングが管理され、このため開発者が、コンポーネントの組み合わせ方を考える必要はありません。開発者は単に使用すればよいだけで、細部は Web アプリケーションフレームワークが自動的に管理します。

また、JSP の作成中に開発者がこれらの限定名を使用することはありません。Web アプリケーションフレームワークが、「コンテキストタグ」と呼ばれるタグを提供します。このタグは、入れ子にされたコンテナとコンポーネントのスコープを定義します。JSP では、開発者はそのスコープ内でローカル名を使用し、その名前は、現在のコンテキストを使用して実行時に自動的にまたトランスペアレントに限定名に変換されます。親ページでビューコンポーネントを自由に組み合わせられるばかりでなく、生成指定フラグメント (JSP フラグメントおよびページレット) も自由に組み合わせることができます。Web アプリケーションフレームワーク開発者には、自由に使える 2 通りのビューコンポーネントの再利用方法があり、それらはいくつかの順列で組み合わせることができます。このことは、単純に他のフレームワークでは不可能です。

効率的なオブジェクト管理

多くのフレームワークは、オブジェクトの割り当てを回避するために効率性とスケーラビリティの向上を意図してアプリケーション内でのオブジェクトの再利用に重点を置いています。しかし現在ではこの手法は間違いであることが有名ないくつかのフォーラムで明らかにされています。このことは JDK 1.0 の頃には真ではなかったか

もしれませんが、現代の JVM では、オブジェクト割り当ては、特にプロセス全体の同期ポイントに比較するときわめて安直なものとなっています。最大のスケーラビリティを得るには、フレームワークはできる限り同時並行のスレッド間の同期を回避する必要があります。

オブジェクトの共有のためにはどんなことでもするフレームワークは、知らない間にそのスケーラビリティを制限しています。また、複雑さが大幅に増し、マルチスレッドに関係するバグを回避するために細心の注意が必要になります。多くの場合、そうしたフレームワークは、しばしばそうした仕事を受ける用意ができていないアプリケーション開発者にマルチスレッドプログラミングの心配の種を課します。おそらくもっと重大な懸念は、フレームワーク上で構築されたアプリケーションが本稼働で使用され、多大な負荷がかけられるまで、そうしたバグが明らかにならないかもしれないという事実です。

こうした理由のため、Web アプリケーションフレームワークでは実際的な手法を採用しています。意味のあるところではオブジェクトを再利用しますが、必要に応じて他のオブジェクトを割り当てることを可能にします。Web アプリケーションフレームワークの共通の要求処理インフラストラクチャは、コンテナが管理する共有オブジェクトインスタンスに依拠しますが、開発者が使用する通常の要求処理中のオブジェクトは必要に応じて割り当てられます。この手法は、複雑さを減らすばかりでなく、Web アプリケーションフレームワークそのものにとっても 1 つのクラス全体で存在する可能性がバグを排除し、アプリケーションのコードについても同じ働きをします。共有データのことで開発者が足踏みする心配をする必要はなく、デバッグはより簡単になります。

Web アプリケーションフレームワークは、この手法が本稼働配備で最大限に効果的であることを証明しています。実際の稼働で、オブジェクト割り当てを原因とする大幅な待ち時間またはメモリーに対する影響なしで、1 秒あたり数百の要求が処理されています。

並列コンテンツのサポート

現代の大部分のフレームワークは、開発者が Java リソースバンドルを利用できるようにすることによって国際化に対応できるようになっています。JSP 作成者は JSP 内の静的コンテンツをカスタムタグに置き換え、それらのタグで、ローカライズされたコンテンツを、プロパティファイルがサポートするリソースバンドルから実行時に取得します。

この手法は有用ではありますが、単独で使用すると、大きな短所があります。中でも大きいのは、JSP ページの作成者が標準的な HTML エディタを使って自然な方法でページを作成できず、プロパティファイルの内容を編集しなければならないという短所です。また、大体において、この手法では、ローカライズされたコンテンツを囲むマークアップに変更がないことを前提にしています。しかし、現実には、マークアップは、対象となるデバイスや言語によってひどく左右されることがあります。このため、いくつかの種類の国際化は、「並列」「コンテンツ」と呼ばれる代わりの手法を利用して最適な対処をしています。

Web アプリケーションフレームワークでは、並列コンテンツに完全に対応しています。並列コンテンツとは、並列な JSP セットを使用することで、セット内のそれぞれの JSP は、特定の言語や対象デバイス、出力マークアップ (たとえば XML、HTML、WML など) 、あるいはそれの任意の組み合わせ用にカスタマイズされています。これら JSP はそれぞれに同じビューコンポーネントを参照し、JSP には、コンテンツおよびマークアップの変更分だけが含まれます。このためアプリケーションは最も適切な JSP を選択して、ユーザーの設定または他の条件に基づいて実行時に生成することができます。

並列コンテンツは、ページのレイアウトが大きな影響を受けることがある西欧およびアジア言語の両方向けのコンテンツをローカライズするとき、あるいは標準的なブラウザやインターネット対応の携帯電話などの異なるタイプのデバイス向けに生成を行うときに非常に優れた働きをします。この利点は、生成時の相違 (場合によって大きな相違がある) を許容する一方で、ビジネスロジックおよびビュー構造の一貫性が、ローカライズされたページにまたがって維持されることです。

フレームワークによっては、JSP とアプリケーションコンポーネント間の静的な関連付けを前提にしていたり、コンポーネントと JSP の関係の宣言指定を使ってページの流れを自動化しようとしたりします。この後者の手法はいくつかの限られたケースでは利点がありますが (ただし、それ以上に重大な多くの欠点がある) 、並列コンテンツの利用に必要な柔軟性を持つことができません。その他、JSP におけるプログラムコンストラクトを強調しているフレームワークもあり、そうしたフレームワークでは、並列コンテンツの使用はきわめて困難です。これらのフレームワークを使用する開発者は、複数の並列 JSP にまたがってプログラムコンストラクトをコピーし、維持する必要があります。Web アプリケーションフレームワークは表示イベントを提供して、JSP からプログラムコンストラクトを締め出しているため、Web アプリケーションフレームワークアプリケーションでは、並列 JSP にまたがって表示ロジックの複製を作成する必要はありません。

Web アプリケーションフレームワークは並列コンテンツに完全に対応し、アプリケーションが開発者の定義した条件に基づいて生成する JSP を実行時に選択することがきわめて容易です。並列 JSP に対するルックアップも開発者が定義するため、アプリケーションに意味のある方法で並列コンテンツを整理することができます。Web アプリケーションフレームワークの並列コンテンツ機能は、リソースバンドルベースの国際化戦略とともに、最大限に柔軟性のある国際化をサポートします。

すぐに使える高レベルの機能

Web アプリケーションフレームワークは、アプリケーションおよびコンポーネントが使用する低レベルのインフラストラクチャばかりでなく、高機能のアプリケーションを短時間に構築することを可能にする高レベルの機能も提供します。

WebAction はそうした機能の 1 つで、開発者が最小限のコードで共通の高度な仕事を行うことを可能にします。たとえば開発者は Next および Previous WebAction を起動して、要求にまたがってデータセットモデル内のデータ行を自動的にページングすることができます。データセットの位置は、WebAction インフラストラクチャに

よって要求にまたがって自動的に管理され、追加のコードは必要ありません。WebAction では、データセットモデルを実装しているあらゆるモデルを利用することができます。

Web アプリケーションフレームワークが提供するもう 1 つの高レベルの機能として、JDBC リソースに対するモデル指向のアクセスを自動的に管理する一群の SQL ベースのモデル実装があります。これらの実装では、SQL の紹介とストアドプロシージャを利用して、RDBMS 内のデータを検出、維持します。このため開発者が JDBC の使用の細部や JDBC ドライバの使用法の矛盾について頭を悩ます必要はまったくありません。当然のことながら、開発者は、必要に応じて Web アプリケーションフレームワークアプリケーションの中から直接 JDBC を利用することもできますが、Web アプリケーションフレームワークコアにこれらの付加価値実装が存在することによって、既成のままでも、非常に短期間に機能的なエンタープライズアプリケーションを構築することができます。

他のフレームワークには、既成のままでも、あるいは他の形態でも、単純にこのレベルの機能が用意されていません。開発者は、Web アプリケーションフレームワークなどの任意のフレームワークでオブジェクトのリレーショナルマッピングツールを利用できますが、最低限、アプリケーションアーキテクチャで複雑なビジネスオブジェクトを使用する意識的な決定が必要になります。これに対し Web アプリケーションフレームワークの SQL ベースのモデルでは、アプリケーションドメインからこうした細部を抽象化して、標準的なモデルインタフェースの背後に置くことができます。

アプリケーションの他の部分が直接 JDBC や SQL に依存することはなく、このためはるかに保守性がよく、一貫性が維持されます。

まとめとして、Web アプリケーションフレームワークには、階層データの表示の開発を劇的に簡略化するツリービューとツリーモデルプリミティブが用意されています。これらのプリミティブは、ルック & フィールに囚われない一群のカスタムタグによって補完され、それらのタグによって、開発者は、ツリーノード別に選択生成される部分に分けて JSP ドキュメントを構造化することができます。タグそのものはマークアップを出力しないため、JSP フラグメントやページレットで使用して、プラグおよびカスタマイズ可能なコンポーネントのルック & フィールを実現することができます。現代の他のフレームワークで、これらのコンポーネントに匹敵する機能を持つものはありません。

ツール対応

他のフレームワークと異なり、Web アプリケーションフレームワークは何もないところから、最終的に GUI アプリケーションビルダーと組み合わせて Web アプリケーションを作成できるところまで使えるように設計されています。現代の他のほぼすべてのフレームワークには、高機能のアプリケーションビルダーとの統合を可能にする機能はありません。正式なフィールドやコンポーネントを定義していないばかりでなく、ページ主体の開発手法を提供していないため、GUI ビルダーで操作するチャンスはわずかしかありません。統合しても、たいていはテンプレートからの一方向のコード生成に制限され、その後は単調な手動のコード編集作業になります。

基本設計にツール対応が組み込まれてはいますが、Web アプリケーションフレームワークは、いくつかの理由からこの機能をまだ提供していません。いずれ GUI アプリケーション開発に重点を置くとしても、その前にフレームワークそのものを誤りがなく、堅牢で、API ベースの操作をできるようにする必要があるためです。開発者は、綿密に定義された API を使用し、非常に高度な技法の利用をはじめとして、必要なあらゆることを行える必要があります。一般に、最初からツールのサポートを重視したフレームワークは、その種の操作だけに偏り、そのインフラストラクチャの土台となる、綿密に設計されていて使い易く、柔軟性のある API を提供できないでいます。このことは、開発者がツール指向の層の下に行って、再利用可能なコンポーネントの構築や高度な方法でのアプリケーションオブジェクトの処理を行えないことを意味します。

また、ツールサポートを重視することは、パフォーマンスの低下を生じる可能性がある方向に設計を導く傾向があります。おそらくパフォーマンスの問題につながる実装手法で最も一般的なのは、共有オブジェクトの使用です。実行時共有オブジェクトは、アプリケーションビルダー指向の設計では一般的ですが、実行時にかなりの同期を必要とするため、最終的にはスケーラビリティが制限されます。開発のスケーラビリティのために、本稼働のスケーラビリティを諦める必要はないはずで、その両方が実現可能にされるべきです。

Web アプリケーションフレームワークでは、最初にインタフェースおよびオブジェクト規約を軸にフレームワークを設計すれば、後の段階で GUI ビルダーのサポートを追加しやすいという見方をしており、実際に、Web アプリケーションフレームワークで現在追求されています。この結果生まれたのが、アプリケーションビルダーを使って開発の生産性を提供するばかりでなく、真の意味でフレームワークをエンタープライズ向けにする、高度な使用に対応するフレームワークです。このことはまた、この工程の初期にツール主体の設計によってスケーラビリティやパフォーマンスが犠牲になることなく、またアプリケーションビルダーのサポートが多少パフォーマンスの低下を必要としても、開発者は、開発と本稼働のスケーラビリティティのどちらにするか意識的な選択ができることを意味します。

まとめとして、通常、アプリケーションビルダー重視のフレームワークでは、追加の、時として標準外の技術を利用することになり、このことは、適切なバージョンであることを保証できない、あるいは特定のプラットフォームで使用できるとしても、その範囲外の追加のライブラリを必要とすることを意味します。こうした他の技術は、プロジェクトチームによって手動で使用できるようにする必要があり、時としてバージョンの衝突のために困難を伴うことになります。この点では、Web アプリケーションフレームワークは、プロジェクトの配備アーキテクチャと衝突する可能性がある追加のライブラリを必要としない方が優れているという見方をしています。

エンタープライズクラスのパフォーマンス

Web アプリケーションフレームワークは、あらゆる同期点を排除するように最適化されています。このため、Web アプリケーションフレームワークで構築されたアプリケーションは、それが動作する J2EE コンテナと同程度にスケーラブルです。犠牲が大きい同期を行う他のフレームワークでは、負荷が大きくなるにつれて、オーバー

ヘッドが幾何級数的に増大したり、待ち時間がより長くなったりする可能性があるのに対し、Web アプリケーションフレームワークは、どのアプリケーション要求にも一定で小さなオーバーヘッドしかもたらしません。

まとめ

Web アプリケーションフレームワークは、現代の他の Web アプリケーションフレームワークには見られない、あるいは並ぶもののない機能を提供します。現存するフレームワークの大多数は、JSP や XML などのさまざまな技術を使ったデータの生成に重点を置いています。実際には、それなりの範囲の開発者ニーズに対処しようとしているフレームワークはわずかしかなく、広範囲のエンタープライズアプリケーション開発ニーズに対処しようとしているのは、Web アプリケーションフレームワークだけです。Web アプリケーションフレームワークは、一貫して企業開発を補完し、企業開発に特有の難題がもつ影響を最小限に抑えることを目的に設計されています。

つまり、Web アプリケーションフレームワークは、以下の方法でエンタープライズフレームワークの基準を満たします。

- 最新技術を結集した実証済みの設計パターンの使用を奨励し、促進することによって、アプリケーションの一貫性を実現する。これによって、開発者の技術をチームやプロジェクト、会社の枠を超えてより簡単に再利用できるようになります。Web アプリケーションフレームワークは理解が容易で実証済みのアプリケーション開発手法を提供するばかりでなく、上級の開発者には、土台の J2EE および Web アプリケーションフレームワークプラットフォームとの低レベルの対話も可能にします。
- Web アプリケーションフレームワークは、すぐに使えるモデル実装から基本的な拡張性機能まで、高低両レベルの機能を提供するため、チームはその要件に応じて適切に機能のバランスをとることができます。
- アプリケーションの保守性を向上させる具体的な手段を提供する。こうした手段としては、退屈なデータシャッフルを無用にする、最新技術を結集した要求送信機構、綿密に定義されたオブジェクト規約による一貫した設計パターンの適用、高度なコンポーネント開発機構、きめの細かいアプリケーションイベントおよびオーバーライドポイント、ページ主体の開発モデルなどがあります。
- すでに精通しているアプリケーション開発概念を利用して、明確で理解が容易な実証済みアプリケーション開発手法を提供することによって、綿密に設計された高性能の Web アプリケーションを作成できるよう未熟な開発者の手引きをする。
- 上級の開発者と土台の J2EE プラットフォームの間に入らず、綿密に定義された拡張性の仕組みを提供することによって、上級開発者の手助けをする。
- エンタープライズクラスの設計パターンや実証済みの高性能、エンタープライズ層のリソースアクセスに対する正式な抽象化およびカプセル化機構を企業の設計者にアピールする。

Web アプリケーションフレームワークは、成熟して堅牢で、安定しており、きわめて優れたパフォーマンスを発揮するフレームワークです。しかし最も重要なことは、Web アプリケーションフレームワークはそうしたことを実証済みであるという点です。毎日数百万のユーザー、数百万ドルの金融取引を支える本稼働のエンタープライズアプリケーションで成功を収めており、企業の開発者や設計者、プロジェクト管理者から等しく絶大な支持を受けています。ここで概説した他のどの理由にもまして、Web アプリケーションフレームワークがエンタープライズの世界ですでに実証済みであるという事実は、Web アプリケーションフレームワークがエンタープライズクラスの Web アプリケーションフレームワークのあらゆる基準を満たしていることを、Web アプリケーションフレームワークを採用している人々に確信させるものです。

第2章

Web アプリケーションフレームワークの設計とアーキテクチャに関してよく受ける質問 (FAQ)

この章では、Web アプリケーションフレームワークを初めてお使いになる方からよく受ける、設計とアーキテクチャに関するいくつかの質問に対する回答をまとめています。

この章に含まれている質問は以下のとおりです。

- Web アプリケーションフレームワークはどのようなユーザーを対象としていますか。
- J2EE がすでにあるのに、なぜ Web アプリケーションフレームワークを使うのですか。
- Web アプリケーションフレームワークはもう 1 つの一社専有の Web アプリケーションフレームワーク (JAPWAF: Just Another Proprietary Web Application Framework) ではないのですか。
- Web アプリケーションフレームワークと他の J2EE フレームワークはどこが違うのですか。
- Web アプリケーションフレームワークには、「表示フィールド」の概念があり、このことは、J2EE Blueprints や私が目にした他の J2EE アーキテクチャと異なります。なぜ、ヘルパー Bean から値を直接引き出さないのでしょうか。
- Web アプリケーションフレームワークでは、EJB の使用は必須ですか。
- Web アプリケーションフレームワークアプリケーションはどのような構造になっていますか。
- 要求の流れと URL 形式はどのように実装されていますか。
- ビュー Bean とセッションまたはエンティティ Bean の関係はどうなっていますか。
- スレッドセーフ (threadsafe) のコーディングを簡単にし、各要求で強制的に Bean が作成、破壊されるようにするために、JSP のスコープを「要求」に設定している場合、パフォーマンスに悪影響はありますか。

Web アプリケーションフレームワークはどのようなユーザーを対象としていますか。

Web アプリケーションフレームワークは、主として、中大規模および巨大規模の Web アプリケーションを構築する J2EE 開発者のニーズに応えることを意図しています。Web アプリケーションフレームワークは堅牢な設計パターンと、等しく堅牢なそれらパターンの実装を組み合わせて エンタープライズ Web アプリケーションの土台を提供します。Web アプリケーションフレームワークは、再利用可能な JavaBean のようなコンポーネントの中核機能を提供するため、簡単に Web アプリケーションに組み込むことが可能な、既成のコンポーネントを提供することを望む開発者にも適しています。これと同じ機能によって、Web アプリケーションフレームワークは垂直型 Web アプリケーションや製品を構築するためのプラットフォームとして非常に適したものになっています。これは、その水平型の拡張機能によって、ユーザーとオリジナル開発者の両方に、既存の垂直型機能を拡張したり、活用したりするための綿密に定義された手段が提供されるためです。

J2EE がすでにあるのに、なぜ Web アプリケーションフレームワークを使うのですか。

J2EE は比較的新しい技術です。非常に有望な技術ではありますが、J2EE 以外の一部 Web 技術が時間をかけて発展させてきた、機能の豊富さや短期開発モデルをまだ実現していません。このことは必ずしも悪いことではありません。標準外のアプリケーション API から自由であることは大きな利点があり、J2EE が提供する自由は、多数の Web 開発業務をより容易かつ迅速にして、保守性に優れた結果をもたらすことができます。

しかしながら、現在の業界は機能豊富な Web アプリケーションを短期間に構築する必要があり、J2EE ではまさしくそうした課題を規定しているものの最低限のレベルでしかないため、依然、実際のほぼあらゆる Web 開発プロジェクト、特にエンタープライズ Web アプリケーションで、J2EE 以外の追加のアプリケーション設計パターンおよび追加の機能の需要があります。Web アプリケーションフレームワークは、この需要に対応したものです。Web アプリケーションフレームワークは、理解しやすく、堅牢でありながら、これまでにない設計の柔軟性と一貫性をもたらします。何よりも、純粋に規格に準拠した J2EE プラットフォームに完全に基づいているため、Web アプリケーションフレームワークの使用によって J2EE を犠牲にすることはありません。両方から恩恵を受けられます。

Web アプリケーションフレームワークはもう 1 つの一社専有の Web アプリケーションフレームワーク (JAPWAF: Just Another Proprietary Web Application Framework) ではないのですか。

いいえ、違います。一社専有の Web アプリケーションフレームワークでは、フレームワークの API に束縛されるばかりでなく (ソースコードが提供されない)、ベンダーの土台のアプリケーションサーバープラットフォームにも束縛されます。そのうちのどちらか一方だけを必要とする場合は、別のベンダーのフレームワークにアプリケーションを移行させることになり、最初からアプリケーションを書き直す必要があります。

Web アプリケーションフレームワークは、これとは異なります。

- Web アプリケーションフレームワークは、完全に J2EE プラットフォームに基づいています。Web アプリケーションフレームワークコアのクラスにはコンテナに固有のコードがありません。このことは、変更なしに任意の J2EE コンテナ内で Web アプリケーションフレームワークを使用できることを意味します。Web アプリケーションフレームワークは、Apache Tomcat や Caucho Resin、Allaire JRun、Sun Java System Application Server、iPlanet Web Server、IBM WebSphere などのコンテナでテスト済みであり、そのすべてで同じ働きをします (コンテナにバグがある場合は除きます。その場合は、少し異なる使い方が必要になることがあります)。J2EE が提供するすべての機能が提供可能で、その結果 J2EE の使用によって得られる利益をすべて享受できます。
- Web アプリケーションフレームワークの完全なソースコードが提供されます。これが必要になる状況は望ましくありませんが、このことは、Web アプリケーションフレームワークのユーザー自身が、土台の設計パターンを含めて、自分の正確なニーズに合わせてフレームワークの隅々まで調査、変更、修正、操作、構成できることを意味します。Web アプリケーションフレームワークの仕組みを自分の目で確かめることを推奨します。より短時間かつ簡単にバグを修正するのに役立つばかりでなく、Web アプリケーションフレームワークそのものも専門的で詳細な検討や議論から恩恵を受けることができます。
- 基本的に Web アプリケーションフレームワークは、完全にインタフェースとオブジェクト規約に基づく設計パターンです。それらインタフェースのデフォルトの実装は提供されますが、その実装の仕組みを好まない場合は、その部分を置き換えることができます。あるいは、自身でインタフェースを実装し直して、Web アプリケーションフレームワークの他の部分にシームレスに統合される新しいオブジェクトを作成することができます。すべて同じオブジェクト規約に従っていますから、そのようにして作成した新しい Web アプリケーションフレームワークオブジェクトも、他の種類の Web アプリケーションフレームワークオブジェクトと対話することができます。任意の一社専有の Web アプリケーションフレームワークでこのことを試してみてください。

Web アプリケーションフレームワークと他の J2EE フレームワークはどこが違うのですか。

Web アプリケーションフレームワークが登場した前後の J2EE フレームワークに関する調査では、一般に他の J2EE フレームワークは、企業の J2EE 開発者のあらゆる範囲のニーズに完全に対処するようになっていないことが判明しています。むしろ、それらのフレームワークは広範囲の企業開発のニーズの限られた部分を解決しようとしているだけであり、このため大規模な現実のエンタープライズ Web アプリケーションの構築に使用すると欠陥のあるものなっています。

おそらく、一般によく見られる最大の欠陥は、JSP の生成とタグライブラリに最大の重点が置かれていることです。そうしたフレームワークとしては最もよく知られているのは、Apache Struts だと思われます。JSP はあらゆる J2EE Web アプリケーションに不可欠な要素であり、タグライブラリは JSP 作成コストを削減する上できわめて重要ですが、それらのことが、アプリケーションの保守性を最大限に高める一方で、開発者の仕事を最小限にしようとするフレームワークの最大の焦点になることはありえません。それらは、ともに現実世界の企業開発できわめて重要です。たとえば、JSP 内のあらゆる種類のプログラム構成は、保守に問題があります。これは、JSP 作成者にアプリケーション機能が明かされること、また潜在的に多数の場所で並列コンテンツによってこの機能の複製を作成する必要があるためです 。さらに、これらのコンテンツが Java コードほどに機能豊富で強力であることはめったになく、比較的一般的だが複雑な状況に対処するために、JSP にスクリプトレットを作成する必要があるというさらに悪い問題につながる可能性があります。Apache Struts では、この種のアプリケーション開発が強調されており、その機能の多くはそうした機能を埋めることを目指しています。

この種の機能は、大きな保守を必要としないか、限られた寿命の小中規模のアプリケーションには、生産性の利点があると考えられますが、企業では、そうしたアプリケーションは一般的ではありません。これに対し Web アプリケーションフレームワークでは、JSP の長所を認識し、保守性あるいはアプリケーション開発者が JSP の生成を細かく制御する能力を損うことなく、それらの長所を活用します。Web アプリケーションフレームワークは、独立してビュー層を生成指定 (JSP) と生成ロジックコンポーネント (「ビュー」コンポーネント) に組み込むことによって、これらの目標を達成します。これらエンティティの組み合わせでは、JSP からプログラム構成が排除される一方 (JSP では、プログラムコンテンツがコンテンツと混在され、保守が難しくなっている)、きめ細かなビュー関係のイベントを使用することによって生成に対してずっと大きな制御を行えるようになります。

他のフレームワークによく見られるもう 1 つの欠陥は、モデル、またはバックエンドコンポーネントのビュー層インタフェースに関する正式な概念が欠けていることです。開発者が拡張性のある Web アプリケーションを短期間に開発するには、アプリケーションデータを提供するとビューコンポーネントと、そのデータを生成するコンポーネントとの間に規約が定義されている必要があります。他のフレームワークでよく見られるのは、そうした規約またはインタフェースの仕様が存在しないため、開発者がビュー層に固有の形式でデータ (具体的なオブジェクトインスタンスなど) を提供するか、バックエンドからビューへのデータの操作のために冗長なコードを記述せ

ざるを得なくなります。ここでもまた、小規模のプロジェクト、あるいはバックエンド層に対する変更を将来必要としないプロジェクトであれば、許容される場合もあるでしょう。

Web アプリケーションフレームワークは、そのモデルインタフェースを介してビューとバックエンド間の正規な規約を設けているため、ビューコンポーネントはバックエンドコンポーネントから完全に独立することができます。この機能によって、開発者は、ビューそのものを変更することなく、ビューに関係するバックエンドをシームレスに変更することができます。このことは、アプリケーションの寿命の初期には、ビューは SQL 照会から直接データを生成でき、アプリケーションのエンタープライズ層が成熟したら、EJB からデータを生成できることを意味します。このとき、ビューコンポーネントが違いを意識することはまったくありません。このため、大部分の他のフレームワークが実際にはビュー / コントローラアーキテクチャしか提供しないのに対し、Web アプリケーションフレームワークは完全なモデル / ビュー / コントローラアーキテクチャを提供します。

まとめとして、一般に他のフレームワークは Web アプリケーションの送信サイクルを単に考慮していないため、アプリケーションコンポーネント間のインターリンクと関係の規定がなく、保守は難しいものになります。この省略は開発者に負担を荷すことになり、その負担はプロジェクトが始まった時点ではすぐには明らかになりません。たとえば多くのフレームワークは発信データ用の構造を提供しますが、受信データ用の構造はほとんど、あるいはまったく提供しません。このため、起動されたコンポーネントは、どのような要求が発生しても受信しているデータが何であるのか確実に知ることができず、暗闇の中で仕事をせざるをえません。

また、同じコンポーネントに複数のアプリケーションパスがあるため、ターゲットコンポーネントが必要とするデータの準備を呼び出し元に任せざるをえません。このことは、オブジェクト間の依存を増大させるため、著しく保守性向上の妨げとなります。一般に、この関係の増大は、ターゲットコンポーネントまたはバックエンドに、手動で入力データをシャッフルする以上のことをしない必要がある低レベルのコントローラロジックの増殖に反映されます。そして、このことは、バックエンドコンポーネントまたはモデルを使ってページを生成するという不均衡な考え方につながる可能性があります。しかも、そのオブジェクトまたはモデルは、以前に生成したページからの要求を直接処理する際に使用されるわけではありません。この不均衡により、開発者にはさらなる負担がかかり、バックエンドコンポーネントを細かく管理し、Web アプリケーションコンテナの実行という低レベルの細部を心配しなければならなくなります。

それに対し Web アプリケーションフレームワークでは、このクラスの機能をコアの設計パターンと実装に組み込んでいます。このため、開発者は、生成されるビューやバックエンドとやりとりされるデータの生成に関係することから完全に解放されます。この結果、開発者から見ると、モデルは要求にステートフル状態を維持し、その実際のステートフル状態のためにアプリケーションに負担を課すこともありません。

要約すると、Web アプリケーションフレームワークは企業開発者のあらゆる範囲のニーズに完全に対処し、1 つの技術あるいはそうしたニーズの一部に焦点をあてることを回避しています。他のフレームワークは、企業開発の 1 つないし 2 つの側面に対

処する狭い手法をとりがちで、現実世界の大規模なエンタープライズ Web アプリケーションの構築に必要な主要な側面のすべてに対処していることはあまりありません。

Web アプリケーションフレームワークには、「表示フィールド」の概念があり、このことは、J2EE Blueprints や私が目にした他の J2EE アーキテクチャと異なります。なぜ、ヘルパー Bean から値を直接引き出さないのでしょうか。

表示フィールドのパラダイムは、より原始的な技法に比べて独特の利点をもたらします。それらの利点を説明する前に、「表示フィールド」のことを頭に描くと、アプリケーション内で「表示フィールド」を使用する理由はないことに注意してください。コンテナと子ビューの仕組みは、埋め込み可能な任意のビューオブジェクトという概念に完全に基づいています。子ビューオブジェクトはショッピングカートのまるごとの表示あるいはアプリケーションメニューのようにきめの粗いこともあれば、個々の表示フィールドのようにきめの細かいこともあります。この柔軟性は、単体モジュールからのアプリケーションの構成ばかりでなく、より従来型の表示フィールド指向の手法を可能にします。手短に言えば、Web アプリケーションフレームワークでは、ヘルパー Bean から値を直接引き出せるだけではありません。

最上位の `ViewBean` インスタンス (「ルートビュー」) はどれもコンテナビューのインスタンスで、任意の組み合わせのサブビューを含むことができ、その一部を、表示フィールドビューにすることができます。タイルビューもまたサブビューであり、それ自体子ビューを含めるほかに、自由に入れ子にすることができます。

このビュー階層は、一般的な Web 層開発者が予想するものより少し複雑です。たとえば、多くのそうした開発者は、関係する JSP の生成に必要な値を取得するためのメソッドのすべてを宣言したヘルパー Bean を単に作成するでしょう。その場合、それらの値のソースは、そのヘルパー Bean のメソッドの内部でカプセル化され、開発者は、*<jsp:setProperty>* タグの中で、送信する要求パラメータを自動的に Bean のフィールドにマッピングする「*」表記を使用します。この方法は単純明快ですが、開発および保守の面でかなり大きな欠点があります。

方法としては Web アプリケーションフレームワークも似ていますが、モデルとビューの厳密な分離を維持する上で、サブビューの使用が非常に重要になります。たとえば、表示フィールドビューはすべて 1 つのモデルに「バインド」されます。表示フィールドビューには、そこに含まれる値の概念はありません。また、すべての表示フィールドは、「バインド」されますが、必ずしもすべてが「データにバインド」されるわけではありません。このモデルは、単なるメモリー上の記憶領域である `DefaultModel` のインスタンスであることもあれば、SQL 照会やストアドプロシー

ジャ、EJB、ビジネスオブジェクト、XML DOM、さらには SOAP プロシージャであることもあります。表示フィールドビューは、データおよびビジネス動作の記憶および管理からは完全に独立しています。

表示フィールドまたは、任意のモデルにバインド可能で、バインド先のモデルの種類の知識なしに機能するという点でモデルに囚われません。このことは、値のソースから値の使用者に値を移動 (いわゆる「データシャッフル」) するためのアプリケーションコードを書く必要がないことを意味します。EJB や JDBC 結果セットなどから値を取得するために、Bean の補助メソッドの中で、ターゲットに固有のコードを書き、また保守する必要がある、簡単なヘルパー Bean のケースと異なり、Web アプリケーションフレームワークでは、無償でこれが手に入ります。要約すれば、表示フィールドは、任意のモデルのシームレスなプラグ化を可能にするために最低限必要な間接化をもたらします。また、表示フィールドと関係するモデルとの対話は非常に一般的な「モデル」インタフェース経由で行われるため、バックエンドデータはそのネーティブの形式で表現することができます。ビュー層だけが必要とするデータ形式のためにデータを操作するというコストのかかる作業は必要ありません。

さらに加えれば、表示フィールドは一般的なヘルパー Bean/taglib 手法に比べて、大幅にプログラミングモデルを改善します。制御ロジックによる直感的な操作を可能にするばかりでなく、型特有の操作や HTML 生成インタフェースを提供します。これらのどれも、他の手法では不可能なことです。

たとえば、一般的な taglib 手法の欠点の 1 つは、HTML 生成プロセスに対する実際の入力がヘルパー Bean にないことです。このため、よくあるようにこの生成を制御する必要がある場合、しばしば、JSP には、スクリプトレットとして、扱いにくい量のアプリケーション関係のロジックとプロパティが発生することになります。たとえば現在のユーザーにフィールドを見る権限がないため、そのフィールドの生成を省略したいと仮定します。一般的な JSP では、開発者はこの表示を回避するためにスクリプトレットを用意して、JSP 内に Java コードを入れることになります。もう 1 つの方法として、ハンドラの機能を強化して、条件変数を調べるなどの、何らかの標準的な仕組みに基づいて表示に条件を付けるようにします。どちらの方法も、問題は、アプリケーション関係のデータと表示関係のデータの一貫性がなくなるか、両者の分離が行われないことです。言い替えれば、簡潔さに欠け、保守性が劣ります。

Web アプリケーションフレームワークでは、両者のうちの優れている方法を提供することによって、この制限を回避しています。たとえばフィールドの表示に条件を付けたり、フィールドの HTML 出力をカスタマイズしたい場合、開発者は最終段階で親ビューに表示イベントハンドラを実装することができます。この場合、そのハンドラは、HTML の生成中に自動的に起動されます。開発者はそのフィールドの表示を省略することも、あるいは HTML 出力を直接操作 (テキストボックスのプレーンテキスト化など) することもできます。あるいは、表示フィールドビュー上で、必要なアクションを支持するメソッドを呼び出すこともできます。この場合、そのアクションは、HTML 生成サブシステムによってフィールドを生成するときに自動的に考慮されます (JSP/taglib の組み合わせ)。

つまり、開発者は、ページコンテンツとともに表示関係のコードを JSP や制御ビジネスロジック、ビジネス指向のモデルに入れることなく、表示フィールドの生成プロセスを補強するか、簡単にそのことを指示することができます。この手法は Java コー

ドを JSP から独立させるばかりでなく、スクリプトレットや他の HTML 生成制御手法に比べて、はるかに強力です。もう 1 つの利点として、一貫性でもはるかに優れています。

表示フィールドはまた、HTML ページがステートフル状態のサーバー側オブジェクトであるかのように、HTML ページを操作することを可能にします。ユーザーが HTML ページ上でボタンまたは HREF をクリックした場合、最終的にはその要求は、そのボタンまたは HREF を生成したビューに返され、そのオブジェクトに対応するイベントハンドラが起動されます。しかし、Web アプリケーションフレームワークでは、イベントハンドラを起動する前に、送信された要求パラメータを使い、すべての表示フィールドとビューを再生成します。この結果、開発者から見ると、ページはステートフル状態を維持し、単にユーザーからのコマンドに応答するだけです。この場合、たいてい、開発者は、イベントハンドラを実装し、要求と更新されたフィールド値に基づく処理を行うことによって、「fat クライアント」アプリケーションで行うのと同じ方法でイベントを処理します。表示フィールドは必ずモデルにバインドされるため、フィールドの値に変化があると、自動的にそのことがモデルに伝達されます。このため、開発者は、従来の「fat クライアント」のようなプログラミングスタイルと正式な MVC のようなプログラミングスタイルとの間で最も生産的な方法を選ぶことができます。

ヘルパー Bean や表示フィールドによる代替手法はともに、HTML ページとやりとりするインタフェースがそれらのオブジェクトによって処理されるという点で、ファサード設計パターンを実装することに注意してください。このインタフェースを介して提供されるデータ値は、複数のバッキング EJB やビジネスオブジェクト、結果セットなどから提供することができ、抽象的な意味で、そのどれもが「モデル」と言うことができます。ただし、従来の ヘルパー Bean がカスタムコードを使ってそれらのモデルを管理して、一貫性や再利用可能性がほとんど、あるいはまったくないのに対し、Web アプリケーションフレームワークでは、それらを 1 つの「モデル」という正式定義に抽象化し、モデルとそのモデルにバインドされたビューとの間に明確な規約を規定しています。

この正式な規約には、いくつかの利点があります。

- ビューの提供するファサードとそのファサードを支える一群のモデルとの間に、簡単に変更することが可能なマッピングを表示フィールドが提供することを可能にします。Web アプリケーションフレームワークには XML ベースの宣言機能が含まれているため、表示フィールドを使って、このファサードに対する宣言の変更を行うことができ、アプリケーション開発を大幅に簡略化するとともに、開発時間を短縮することができます。
- このファサードは非常に簡単に変更できるため、バッキングモデルに対する変更を、きわめて容易に、また宣言サポート付き、再コンパイルなしでアプリケーションのビュー部分に伝達することができます。この手法と従来の ヘルパー Bean の手法とを比較してみてください。ヘルパー Bean 手法では、開発者は、ヘルパー Bean に補助メソッドとミュテータメソッドを追加するか、バッキングモデルと値をやりとりするために必要なコードを変更するか、その両方を行う必要があります。こうしたきめ細かな変更と再コンパイルが必要になるため、開発時間や生産性は低下し、オブジェクトの再利用も難しくなります。

Web アプリケーションフレームワークでは、EJB の使用は必須ですか。

いいえ、必須ではありません。他のあらゆる J2EE アプリケーション同様、EJB への参照を入手し、Web アプリケーションフレームワークアプリケーションの中からその参照を直接使用することができます。EJB は J2EE のコンポーネントではありますが、必須コンポーネントではありません。また、EJB はアプリケーションのアーキテクチャを比較的複雑にし、現状では、いくつか大きな欠点があります。独特の問題が数多くあり、アプリケーション設計と実装の多くの分野にかなりの追加投資が必要になるため、多くの Web 開発者が EJB ベースのアーキテクチャに移行する用意ができていません。このため、EJB の使用は必須となっていません。

ただし、Web アプリケーションフレームワークで、EJB を使用することはできますし、使用を容易にしています。Web アプリケーションフレームワークには、柔軟でプラグ可能なモデル / ビューアーキテクチャに基づく価値ある機能が用意されています。それらの機能は Web および EJB 指向の両方のアプリケーションを補完します。EJB の直接使用に代わる方法として、たとえば開発者は、EJB の支える、または EJB が直接実装しているモデルを利用することができます。Web アプリケーションフレームワークでは、EJB の組み込みは、その種のあらゆるモデルに対するのと同様シームレスであり、自動的なデータバインドおよびその他高レベルの機能を提供します。

EJB に取り組む最初の手段として、Web アプリケーションフレームワークには、標準の「モデル」インタフェースを任意の JavaBean プロパティにマップする `BeanAdapterModel` というクラスが含まれています。顧客 EJB がある場合は、このアダプタモデル内でその EJB をラップし、追加コードなしに、表示フィールドによって、そのプロパティ / 値を直接利用することができます。アダプタを使用して、表示時には、顧客 Bean から値が生成され、送信時には、その値が顧客 Bean に戻されます。このクラスを使い、ローカルのビジネスオブジェクトへのアクセスを同様にラップすることもできます。このときアダプタは、カプセル化したオブジェクトがリモートまたはローカルのどちらであるかを意識しません。

Web アプリケーションフレームワークアプリケーションはどのような構造になっていますか。

Web アプリケーションフレームワークアプリケーションは、1 つ以上の「モジュール」から構成される完全に独立したエンティティです。1 つのモジュールは、アプリケーション全体のうちの 1 つの機能スライスまたは論理的なまとまりです。アプリケーションには少なくとも 1 つのモジュールですが、他のモジュールは任意で、いつでも追加することができます。

1 つの Web アプリケーションフレームワークアプリケーションは、モジュール専用のすべてのサーブレットのもとになるベースクラスである、「アプリケーションサーブレット」と呼ばれるものを定義します。アプリケーションのクライアントは、モ

ジュールサーブレットにだけアクセスします。アプリケーションサーブレットは、モジュールサーブレットの共通のベースクラスとしてのみ機能し、1 つのクラスにアプリケーションレベルの共通のイベントハンドラを統合できるようにします。モジュールサーブレットは、これらのイベントのモジュール専用化を可能にします。これらのサーブレットは一体となって、各 Web アプリケーションフレームワークアプリケーションの要求処理インフラストラクチャを形成します。

各モジュールはアプリケーションのサブパッケージに相当し、そのモジュール専用の最小限のサーブレットインフラストラクチャを含むほか、1 つ以上の論理ページを含みます。各モジュールにはまた、サポートモジュールと他の非 Web アプリケーションフレームワーククラス (当然、モジュール内のクラスは通常通りモジュール外のクラスを使用可能) が含まれることがあります。

モジュール内の 1 つの論理ページは、最低限 1 つの JSP と 1 つの ViewBean から構成されます。ViewBean は、対応する JSP のヘルパー Bean であり、Web アプリケーションフレームワークのタグライブラリとの組み合わせで、表示 / アプリケーションイベントインフラストラクチャを提供します。各 ViewBean には、組み立てることによって、わずかな時間で本格的なデータバインドページを作成することを可能にする、再利用可能な子ビューオブジェクトからなる任意の階層が含まれます。

要求の流れと URL 形式はどのように実装されていますか。

当初、あらゆる要求はコントローラサーブレット (モジュール 1 つにつき 1 つで、アプリケーションに複数存在) によって処理され、このサーブレットの URL は開発者が選択しておきます。コントローラサーブレットは要求別のコントローラオブジェクト (ViewBean) に要求を送信し、最終的には別のリソース (JSP か ViewBean、他の Web リソースのいずれか) に転送されます。サーブレットのソースコードとその送信機構は完全に開発者の管理下にあり、開発者には、十分なコメントの入ったソースを読むことによって、この機構の細部を学ぶことを推奨します。

ビュー Bean とセッションまたはエンティティ Bean の関係はどうなっていますか。

直接の関係はありません。ビュー Bean は JSP のワーカーまたはヘルパー Bean (「usebean」) です。各ビュー Bean はそのピア JSP 用の中心的なサポート機能として機能します。セッションおよびエンティティ Bean には、必要に応じてビュー Bean または関係するモデルかビューの中からアクセスすることができます。

スレッドセーフ (threadsafe) のコーディングを簡単にし、各要求で強制的に Bean が作成、破壊されるようにするために、JSP のスコープを「要求」に設定している場合、パフォーマンスに悪影響はありますか。

一般に、持続アプリケーションオブジェクトをサポートするオーバーヘッドは、現在の手法に存在するオーバーヘッドより大きくなります。テストでは、一社専有の非 J2EE コンテナが以前に行っていたことばかりでなく、Web アプリケーションに関係する一般的な動作と比べると、現在の手法からのオーバーヘッドは取るに足りないことが明らかになっています。本質的に、現在の JVM におけるオブジェクトの割り当ては、この手法がもたらすメリットを正当化するのに十分なコストの安い処理です。ただし、このことはそうした手法の意味が無視されてきたことを意味するものではありません。

非 J2EE コンテナの中には効率性の観点から持続アプリケーションオブジェクトを提供していたものがありましたが、そうしたオブジェクトはどのクライアントにもステートフルではありません。このため、各クライアントのステートフル情報は、要求のたびに再生成する必要がありました。注意深く、できる限りオーバーヘッドの小さい、同等のアプリケーションオブジェクトが設計されたのはこのためです。多くの場合このことは、どの場合にも要求のたび有用なステートフル情報を再作成しなければならないことと同じです。Web アプリケーションフレームワークでは、そうしたオブジェクトに対して最適化された形でのアクセスをサポートし、このため、必要なときにのみ作成すればよいだけです。このため、一般に Web アプリケーションフレームワークベースのアプリケーションは、処理およびメモリー消費の両面で非 J2EE アプリケーションよりも効率性に優れ、たいていの場合、この差はかなりの開きがあります。

必ずしも、持続アプリケーションオブジェクトを有用な方法で利用できるインフラストラクチャを設計する必要があるだけではありません。Sun Java System Application Server および他の高スケーラブルなサーブレットコンテナは複数の JVM を利用し、必ずしも、ユーザーの要求を同じアドレス空間に戻す必要がありません。このことは、どのような場合にも要求ごとに持続アプリケーションオブジェクトを初期化し直す必要があることを意味します。これは、単にオブジェクトを再作成するのと少なくとも同等のオーバーヘッドです。セッションにアプリケーションオブジェクトを置くことは、きわめてコストの高い処理になり、最低限デシリアライズ化が必要になるため、実用的な選択肢にはなりません (デシリアライズ化は単純なオブジェクト割り当てよりコスト上高価になる)。

持続アプリケーションオブジェクトをサポートするための機能層を Web アプリケーションフレームワークに追加すると、大幅に開発者が土台のコンテナから隔てられます。一般にこのことは、可能な限り標準の J2EE に近づけるという設計原則に反します。

こうした異議を唱える過程で、将来的に、潜在的な利点が欠点に勝る場合は、持続アプリケーションオブジェクトを使用する機能が提供されるようになることもあり得ます。しかし、そうした機能が存在したとしても、狭い範囲のアプリケーションにしか適用できず、一般的な Web アプリケーションフレームワークアプリケーションには役立たないと考えられます。

索引

あ

い

え

か

き

こ

さ

す

せ