

Oracleホワイト・ペーパー  
2010年10月

# JavaServer Faces 2.0の概要および Oracle ADF FacesとOracle JDeveloper 11gで の導入計画

はじめに.....	5
JSF 2.0の新機能.....	5
ページを定義するための新しい方法 : Facelets VDL.....	6
テンプレート.....	6
複合コンポーネント.....	6
リソース処理.....	7
クライアント動作.....	7
Ajax統合.....	7
GETリクエストとブックマーキングのサポート.....	8
システム・イベント.....	8
JSFアノテーション.....	9
faces-configの順序.....	10
新しいスコープ.....	10
新しい暗黙的ELオブジェクト.....	10
ナビゲーション.....	10
プロジェクト・ステージの構成.....	11
JSF 2.0とADF Facesの比較.....	11
Ajax.....	11
部分的な状態保存.....	12
Facelets.....	12
動作.....	12
リソース処理.....	13
ビュー・スコープ.....	13
複合コンポーネント.....	13
JSFページのブックマーキング.....	13

暗黙ナビゲーション .....	14
アノテーション.....	14
エラー処理.....	14
検証.....	14
ネーミング・コンテナのセパレータ .....	14
プロジェクト・ステージ .....	15
ADF Faces JSF 2.0の採用 : Q & A.....	15
Oracle JDeveloperリリース固有の質問 .....	15
ADF Faces固有の質問.....	16
ADFバインディング固有の質問 .....	20
ADFコントローラ固有の質問.....	20
結論.....	20

## 免責事項

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料にしないでください。オラクルの製品に関して記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。

## はじめに

JavaServer Faces (JSF) 2.0は、Java Enterprise Edition (Java EE) 6での新しい、標準のWebユーザー・インタフェース・テクノロジーです。これは、同じテクノロジーのバージョン1.2に対する更新です。

このJSF標準への改訂では、リッチ・インターネット・アプリケーション (RIA) の開発者のための2つの重要なテーマが導入されています。1つ目は、Asynchronous JavaScript with XML (Ajax) のJavaServer Facesリクエスト・ライフ・サイクルへのネイティブな統合であり、2つ目は、デフォルトのビュー宣言言語のJava Server Pages (JSP) からFaceletsへの変更です。さらに、その他の新機能として、複合コンポーネントを構築するためのサポート、ナビゲーションでの追加の範囲と拡張、エラー処理、アノテーション・ベースのマネージド・ビーン構成などがあります。

次リリースとなるOracle JDeveloper 11g (11.1.2) で、オラクルは、JSF 2.0に対するサポートを開発環境とADF Faces JSFコンポーネント・セットの両方に導入することを計画しています。このホワイト・ペーパーは、JavaServer Faces 2.0標準について簡単に紹介することにより、開発者がベースラインのJSF 1.2とJSF 2.0の違い、およびADF Facesの一部と標準のバージョン2.0の間に存在する機能的な重なるの両方を理解する手助けとなることを目的としています。

このホワイト・ペーパーでは、ADFプラットフォームの計画された採用およびJavaServer Faces 2.0との統合に向けたロードマップの概要について説明します。この初期のJSF 2.0統合のターゲット・リリースは、現在2011暦年(カレンダー・イヤー 2011年)に予定されているOracle JDeveloperバージョン 11.1.2です。

## JSF 2.0の新機能

より大きなJava Enterprise Edition 6仕様の一部であるJSF 2.0標準は、SUNのリーダーシップの下に、Javaコミュニティ・プロセス (JCP) 内のJSR-314エキスパート・グループ (EG) によって開発されました。このEGは、Oracle ADF FacesおよびApache Trinidadフレームワークの指導的な開発者を含む、JSFコミュニティ内のすべての主要関係者からのメンバーで構成されています。

JSF 2.0の新機能の多くは、JSF 1.2仕様の制限を超えて展開された1つ以上の既存のJSFフレームワークまたはコンポーネント・セットに等価な実装があります。この点を考慮に入れると、JSF 2.0は、これらの拡張機能のもっとも優れた部分の標準化と言えます。さらには、たとえば、Ruby on Railsフレームワークから借用したアイデアである新しいフラッシュ・メモリ・スコープなどの最新のアイデアのために、JSFを超えた世界に踏み込むことも躊躇してきませんでした。

## ページを定義するための新しい方法：Facelets VDL

JSF 2.0の新機能であるビュー宣言言語（VDL）APIは、JSFランタイムと、JSFコンポーネント・タグをホストするために使用されるビュー・テクノロジー間のコントラクトを定義します。JSFのバージョン2.0の前は、実行時にJSFコンポーネント・ツリーを生成するためのデフォルトの方法としてJava Server Pages（JSP）が使用されていました。残念なことに、単純な解析-コンパイル-レンダリング・プロセスであるJSPリクエスト・ライフ・サイクルは、より複雑なJSFリクエスト・ライフ・サイクルと統合されていません。一般には、このライフ・サイクルの不一致のため、JSP内で定義されたJSFビューのコンテキストでHTMLタグやJSTLタグを使用していた開発者は、ページ・レンダリングでの予測できない結果に対処する必要がありました。

JSF 2.0では、新しいデフォルトのビュー宣言言語としてFaceletsが導入されています。Faceletsの導入はJSPの欠点に対処し、開発者のページのデザインや再利用の体験を向上させるものです。実行時に中間のサブレットにコンパイルされるJSPドキュメントとは異なり、Faceletsはこの不必要なオーバーヘッドを発生させず、JSFコンポーネント・ツリーを直接構築します。これにより、コンポーネント・ツリーの作成やページ・レンダリング・プロセスのパフォーマンスがはるかに向上します。

FaceletsベースのJSFページを操作する場合、JSF開発者にとってのもっとも明らかな変更はファイル拡張子です。ファイル拡張子は、デフォルトでは、使い慣れた".jsp"や".jspx"ではなく".jsf"です。Faceletsドキュメントのコンテンツは、レイアウト、JSFコンポーネントの参照、式言語を定義したHTMLマークアップの貢献と、外部コンテンツを含めたり、テンプレートを参照したり、パラメータを定義したりするための一連のFaceletsタグです。

## テンプレート

JSF 2.0のFaceletsは、`ui:insert`タグで識別される、JSFの名前付きのファセットに似た名前付き領域を使用したテンプレート化機能を提供します。消費する側のページでは、`ui:composition`タグを使用して、テンプレートのFaceletsソース・ファイルを参照します。

## 複合コンポーネント

JSF 2.0の複合コンポーネントは、アプリケーション固有のリソース・ディレクトリ内に保存されたFaceletsファイルです。複合コンポーネント定義は、HTMLマークアップとJSFコンポーネントで構成されます。消費する側のページには、コンポーネント・タグ名として使用されているコンポーネント・ファイル名を使用して複合コンポーネント・ファイルをホストしているフォルダを指す名前空間の参照が含まれています。複合コンポーネントのインタフェース定義では、コンシューマが検出するコンポーネントの属性、ファセット、およびイベント・リスナーを定義します。

カスタム複合コンポーネントを作成するためのこのファイル・ベースのアプローチは、以前のバージョンの仕様を使用してJSFコンポーネントを構築する非常に技術的なタスクに比べてはるかに簡単です。ADF Facesの宣言的なコンポーネントのメカニズムに精通していれば、すぐにその類似点に気付くはずで

## リソース処理

JSF 2.0は最終的に、コンポーネント開発者が、Faces ServletによってトリガーされたJavaServer Faces リクエスト・ライフ・サイクルの外部にある画像、CSS、JavaScriptなどのリソースを参照するためのメカニズムを提供します。JSF 2.0での標準化の前、この特定のタスクは、異なるコンポーネント・フレームワークによるさまざまな方法で管理されていました。

JSF 2.0では、リソース処理は、あるコンポーネントによって使用されている外部リソースがWebプロジェクトのroot/resourcesフォルダ内か、またはJava Archive (JAR) ライブラリのMETA-INF/resourcesフォルダ内のクラスパス上のどちらかに配置されている可能性があることを指定する、特定のパッケージ化構造を通して実装されます。

## クライアント動作

サーバー側の検証と値の変換を実行するために、JavaServer Faces 1.2の動作タグがUIコンポーネントに追加されました。JSF 2.0では、UIコンポーネントのためにクライアント側のスクリプトを宣言的に実行するための手段として、クライアント動作のアイデアが導入されています。クライアント動作によって、ClientBehaviorインタフェースが実装されます。クライアント動作は、FacesBehaviorアノテーションを使用して、またはfaces-config.xml内のエントリを使用してJSFに登録されます。動作を使用すると、コンポーネントまたはそのレンダラは、そのクライアント上で特定のイベントが発生したときに関連付けられたJavaScriptが確実にページ出力に追加され、実行されるようになります。JSFのリファレンス実装によって提供される事前定義されたクライアント動作タグの例として、Ajaxリクエストを発行するために使用されるf:ajaxタグがあります。

## Ajax統合

リソース管理と同様に、JavaServer FacesのAjaxも新機能ではありませんが、これまではJSFコンポーネント・プロバイダごとに異なった方法で実装されてきました。これらの異なる実装によって、同じWebユーザー・インタフェース内にコンポーネント・セットが混在した場合に多くの競合が発生しました。問題は、さまざまなコンポーネント・フレームワークによるAjaxリクエストに応答してブラウザDOMが使用され、操作される方法が異なっている点にありました。

JSF 2.0では、jsf.ajaxパッケージによって、開発者がクライアント側のコンポーネント・イベントに응答してAjaxリクエストを起動するために使用できる新しいJavaScript APIが提供されます。Ajax APIを使用する場合、開発者は、イベント・ソース、実行するアクション、正常な実行に응答してリフレッシュするコンポーネント、コールバックを処理する関数、およびエラー・ハンドラを定義した引数を指定します。ただし、このAjax APIは、おもにコンポーネント開発者のニーズに適した低レベルのAPIです。このAjax APIを簡素化し、コンポーネント・コンシューマによる宣言的な使用を可能にするために、JSF 2.0には、先に説明したf:ajaxクライアント動作タグが用意されています。開発者はこのタグを使用して、イベントが発生したときに起動するクライアント関数、エラー・ハンドラ、およびリフレッシュするコンポーネントを宣言的に指定します。

## GETリクエストとブックマーキングのサポート

JSFでのデフォルトのナビゲーションは、リクエストURLにリクエスト・パラメータを追加することなく機能するポストバック（HTTP POST）を使用して実行されます。ただし、ポストバックの使用は、ページをブックマーク可能にしたい開発者に常に課題を提起してきました。このシナリオでは、ページの状態を再構成できるように、ブックマークされたURL内のリクエスト・パラメータを取得することが有効です。その結果、JSF 2.0の前にはJSFアプリケーション内でのこのブックマーキング・ユースケースは実現が困難であり、多くの場合はまったく不可能でした。

JSF 2.0では、POSTに加えてGETリクエストに対するサポートが追加されました。GETリクエストを機能させるには、開発者は、受信したURLリクエスト・パラメータをマネージド・ビーン・メソッドにマッピングする新しいタグをページに追加する必要があります。それにより、このマネージド・ビーン・メソッドを使用して、ブックマークされたページの目的の状態を再作成できます。この目的のために提供された新しいタグが、`f:metadata`と`f:viewParam`です。

このメカニズムを補完してGETリクエストで状態を復元するために、JSF 2.0ではまた、開発者が`h:button`および`h:link`コンポーネントから、POSTではなくGETを使用するページ・ナビゲーション・イベントを発行することもできます。リクエスト・パラメータの名前と値は、既存の`f:param`子タグを使用してGET URLに追加されます。このようにして、必要なすべての状態情報が含まれたURLを構成できます。

### PreRenderViewEvent

関連した機能として、開発者はページ上の`f:metadata`タグで囲まれた`f:event`タグを使用して、リクエスト・パラメータが適用された後で、かつ要求したビューがレンダリングされる前に、起動されるページの`PreRenderViewEvent`をリスニングして応答することができます。開発者は、そのビューが表示される前にページ・コンテキストを準備するために、マネージド・ビーン内のイベント・リスナーを定義できます。また、リクエスト・パラメータの情報を使用してプログラマ的に要求したビューから離れてナビゲートすることもできます。

### システム・イベント

フレームワークがユーザーのUIインタラクションやリクエスト・ライフ・サイクル処理にตอบสนองしてブロードキャストするコンポーネントおよびフェーズ・イベントに加えて、JSF 2.0ではまた、コンポーネントまたはアプリケーション・レベルで処理できる一連のシステム・イベントも提供されます。

JSFページ作成者は、マネージド・ビーンから、または親のUIコンポーネント・タグの子として追加できる新しい`f:event`タグを宣言的に使用して、Javaでこれらのシステム・イベントのリスナーを登録できます。開発者がリスニングできるデフォルトのシステム・イベントは次のとおりです。

- `preRenderComponent`
- `postAddToView`
- `preValidate`
- `postValidate`

これらのシステム・イベントは、JSF 1.2でサポートされている既存のライフ・サイクル・イベントに似ているように見えますが、実際には、既存のフェーズ・イベントに比べてはるかにきめ細かなアクセスが開発者に提供されます。

これらのイベントを処理するために使用されるマネージド・ビーン・メソッドは、タイプ `ComponentSystemEvent` (アクション・リスナーや値変更リスナーに使用されるのと基本的に同じAPI) の単一引数を受け付ける署名を備えている必要があります。

## JSFアノテーション

Java Enterprise Editionの仕様全体を通して提起されている主要なテーマの1つに、使用されるメタデータ構成ファイルの数の削減があります。これはおもに、Javaアノテーションの使用を通して管理されます。JSF 2.0では、この原則を採用することで、開発者がカスタムJSFコンポーネントの開発時と、JSFアプリケーションの構築時の両方でアノテーションを使用できるようになりました。たとえば、マネージド・ビーンは、`faces-config.xml`ファイル内で排他的に定義する必要がなくなり、単に目的のJavaコードにアノテーションを追加するだけで十分になりました。

マネージド・ビーンを定義するには、開発者は`@ManagedBean`アノテーションを追加し、それを`@RequestScope`、`@ViewScope`、`@SessionScope`、`@ApplicationScope`のいずれかのスコープ・アノテーションを使用して改良します。そのアノテーション内でマネージド・ビーンの名前が明示的に指定されていない場合は、標準のキャメル・ケース内のJavaクラスの名前が使用されます。管理プロパティも同様の方法で定義できます。

JSF 2.0固有のアノテーションに加えて、`@PostConstruct`や`@PreDestroy`などの、サーブレット・ライフ・サイクルのアノテーションもサポートされます。

同じ方法でコンポーネント開発者の作業を簡素化し、コンポーネント定義でのメタデータ・アーチファクトの必要性をなくすために、それ以上のアノテーションが存在します。この目的には、次のアノテーションが使用できます。

- `@FacesComponent`
- `@FacesRenderer`
- `@FacesConverter`
- `@FacesValidator`
- `@FacesBehavior`

**注:** `faces-config.xml`ファイル内の`faces-config`要素の`metadata-complete`属性が`true`に設定されている場合は、特定のJSFアプリケーションに対してアノテーションの使用を無効にできます。

## faces-configの順序

2.0より前のJSFのリリースでは、クラスパスに、JARファイルからのfaces-config.xmlファイルのロード順序を制御するためのメカニズムがありませんでした。その結果、開発者はフェーズ・リスナーやビュー・ハンドラが起動されたり、ロードされたりする順序を制御できませんでした。JSF 2.0では、この制限がなくなり、複数の構成ファイルに対するロードを調整できるようになりました。

## 新しいスコープ

JSF 2.0では、既存のアプリケーション、セッション、およびリクエスト・スコープに、フラッシュとビューという2つの新しいスコープが追加されました。これらの新しいスコープは、Ajaxのページ・リフレッシュや部分送信に対するより優れたサポートを提供します。

ビュー・スコープ内のオブジェクトのライフタイムは、ビューのロードから、ナビゲーション中のユーザーが別のビューに移るまで持続します。

ビュー・スコープ内のオブジェクトを参照するには、開発者は"viewScope"の識別子が接頭辞として付いたEL式を使用するか、またはJavaでは、現在のUIViewRootオブジェクトでgetViewMap()を呼び出します。オブジェクトのライフタイムの点から見ると、ビュー・スコープはリクエスト・スコープとセッション・スコープの間に存在すると見なすことができます。

フラッシュ・スコープを使用すると、開発者は移行時にオブジェクトをあるビューから次のビューに渡すことができます。ビュー・スコープとは異なり、フラッシュ・スコープはリダイレクトやGETリクエストがあっても存続し、新しいビューがレンダリングされた場合にのみクリアされます。フラッシュ・スコープは、"flash"接頭辞を使用してELからアクセスできます。また、Javaからは、JSF ExternalContextオブジェクト上で公開されている場合にアクセスできます。

## 新しい暗黙的ELオブジェクト

JavaServer Faces 2.0では、式言語からアクセスできる新しい暗黙的オブジェクトが導入されています。これらのうちの2つは、すでに新しいメモリ・スコープとの関連で示されました。新しいオブジェクトは次のとおりです。

- component - 現在のUIComponentインスタンスへのアクセスを提供するオブジェクト
- cc - 現在のカスタム複合コンポーネントにアクセスするためのオブジェクト
- flash - 新しいフラッシュ・スコープを表すオブジェクト
- viewScope - 新しいビュー・スコープを参照するオブジェクト
- resource - リソース・ハンドラにアクセスするためのオブジェクト

## ナビゲーション

ナビゲーションには、JSFフレームワーク内のすべての領域のうちでもっとも大きな変更の1つが加えられました。2.0より前のJSFのリリースでは、ナビゲーションは純粹に、アプリケーションのfaces-config.xmlファイル内の適切なナビゲーション・ケース定義を識別するために使用されるアクションの結果に基づいていました。

JSF 2.0で追加された新しい暗黙ナビゲーション機能を使用すると、開発者は構成ファイル内のナビゲーション・ケースを必要とせずに、あるビューへの直接のナビゲーションを実行できます。暗黙ナビゲーションは、アクションの結果に一致するナビゲーション・ルールが見つからない場合に試行されます。

faces-config.xml内の定義されたナビゲーション・ケースを使用した明示的なナビゲーションと暗黙ナビゲーションの両方が失敗した場合は、以前のバージョンと同様に、viewIdは変更されず、ソース・ページが再表示されます。

明示的なナビゲーションに追加されたもう1つの新機能が条件付きナビゲーションです。この機能のために、エキスパート・グループはfaces-config.xmlのボキャブラリに、開発者がナビゲーション・ケースの一部として条件を定義できる新しいif要素を追加しました。この条件では、式言語を使用して、最終的なナビゲーションを管理するブール値を生成します。JSFの以前のリリースでは、条件を評価し、その結果として適切なナビゲーション・ケース文字列を返すにはマネージド・ビーンを使用する必要がありました。

JSF 2.0でのナビゲーションへの最後の拡張は、プリエンティブ・ナビゲーションです。これを使用すると、開発者は、faces-config.xmlファイルで定義されているナビゲーション・ケースに対するナビゲーション・ターゲットをプログラマ的に決定できます。

### プロジェクト・ステージの構成

最後に、JSF 2.0では、コンテキスト初期化パラメータとしてjavax.faces.application.ProjectStageが導入されています。このパラメータは、JSFアプリケーションを実行するための環境を設定し、その有効な値はDevelopment、UnitTest、SystemTest、またはProductionです。

このパラメータは、Application.getProjectStage()メソッドを使用してJavaからアクセス可能であり、JSFコンポーネントやアプリケーションの開発者がコードをProjectStage設定に基づいて異なった方法で動作するように構成できるようにします。たとえば、開発環境内で実行されているアプリケーションには、本番環境内で実行されている場合より詳細なエラー・メッセージを表示させることができます。

## JSF 2.0とADF Facesの比較

JavaServer Faces 2.0の新機能のいくつかは、Oracle ADF Facesに対応する機能があります。

### Ajax

JSF 2.0では、アーキテクチャ内のさまざまなレベルでAjaxがサポートされています。これには、JavaScript APIであるjsf.ajax.request、f:ajaxタグによるFaceletsでの宣言的なサポートのほか、PartialViewContextなどのサーバー側のさまざまなAPIが含まれます。

JSF 2.0では、完全なページ・リフレッシュの代わりに、クライアントに部分的なレスポンスを送信できます。ADF Faces内の等価な機能は、部分的なページ・レンダリング機能を使用して提供されません。ADF Facesと新しいJSF標準は、次の点で異なります。

- Oracle ADFでは、ADF Facesコンポーネント・フレームワークを使用して、部分ページ・リフレッシュのためのコンポーネントをモデル・レイヤー上のサーバー側のデータ変更に応答して動的に登録します。この自動的な部分ページ・リフレッシュ動作は、JSF 2.0では使用できません。
- Mojarra JSF 2.0のリファレンス実装 (RI) ではiframeベースのAjaxリクエストをサポートしていないため、ファイル・アップロード・コンポーネントがなく、multipart/form-dataリクエストを処理する機能もありません。ADF Facesはこの機能を提供しており、JSFによって標準実装が提供されるまで、独自の実装で引き続き提供します。
- ADF Facesの部分的なリクエスト呼出しは、クライアント側の検証と統合されています。

今後のADF Facesに対する計画は、JSF 2.0のネイティブなAjax APIとADF Facesの部分ページ・リフレッシュ実装の両方をサポートすることです。

### 部分的な状態保存

状態の保存と管理のオーバーヘッドを削減するために、JSF 2.0では、新機能として部分的な状態保存が導入されています。新しいJSF 2.0の実装が成熟する一方で、ADF Facesでは、下位互換性を引き続き提供するために既存の状態保存実装を利用します。

### Facelets

ADF Facesは、JSF 1.2からFaceletsをサポートしてきました。このサポートは、新しい、標準のFacelets APIに合わせるように更新されます。さらに、以前はJSPドキュメントに対してのみ存在していたMDSサポートが、Faceletsに対しても使用できるようになる予定です。

ADF Facesは引き続き、JSPとFaceletsの両方をサポートします。ただし、JSF 2.0の多くの機能がJSPに対して使用できないため、Faceletsが推奨されるビュー宣言言語になります。

### 動作

ADF Facesのクライアント動作機能が、Oracle JDeveloper 11g Release 2では、JSF 2.0のクライアント動作を確実に使用できるように拡張されました。将来のリリースでは、ADF Facesの内部実装を新しいJSF ClientBehavior APIに合わせる予定です。

JSF RIと比較して、ADF Facesのクライアント動作の実装は、イベントがDOMではなく、コンポーネント・レベルで報告されるという点で異なります。イベントは関連するリスナーにブロードキャストされ、ADF Faces固有のコンポーネント・イベントと、JSF 2.0の動作で対応するネイティブなDOMイベントを含んでいます。ADF Facesは、引き続きコンポーネント・レベルのJavaScriptイベントをサポートする一方で、JSF 2.0の動作タグに対するサポートも拡張していきます。

## リソース処理

リソースのロードを効率的に処理するために、ADF Facesでは、`af:resource`タグを使用してページ・ヘッダーにスタイルとスクリプト・ソースを追加します。ADF Facesでは、独自のリソース処理のために、引き続きTrinidad ResourceServletと`af:resource`タグを使用します。また、アプリケーション開発者は、ADF Facesアプリケーション内でJSFリソースの新しいロード・メカニズムを利用することもできます。

## ビュー・スコープ

JSF 2.0は、特定のビューが存在する期間を示す値を保持する、新しいメモリ・スコープであるビュー・スコープを提供します。ADFコントローラもまた、同じ名前のスコープを提供しています。

この2つのビュー・スコープの違いは、JSF 2.0では、ページ・リフレッシュまたはそのビューへのリダイレクトで空になる、UIViewRoot上のマップ内に情報を格納する点にあります。

ADFコントローラのビュー・スコープは、ユーザーが新しいビューにナビゲートして、現在表示されているビューを破棄したときにリフレッシュされます。そのため、ADFコントローラのビュー・スコープ内に格納されているデータは、リフレッシュや同じビューへのリダイレクトがあっても継続します。

ADF Facesは、JSF 2.0を利用して、UIViewRoot上で定義された新しいビュー・スコープをサポートします。既存のADFcのビュー・スコープのより長い期間を保持するために、JSF 2.0のビュー・スコープへのすべてのリクエストが既存のViewScopeプロバイダに透過的に委任されます。

## 複合コンポーネント

JSF 2.0の複合コンポーネントは、ADF Facesの宣言的なコンポーネントとページ・テンプレートに相当します。ただし、JSF 2.0の複合コンポーネントとは異なり、ADF Facesコンポーネントは独自のバックギンク・ビーン・スコープを持つことができます。さらに、JSF 2.0の複合コンポーネントがFacelets VDLに制限されるのに対して、ADF Facesの宣言的なコンポーネントはFaceletsとJSP VDLの両方をサポートします。宣言的なコンポーネントが持つ利点のため、ADF Facesでは、宣言的なコンポーネントを引き続きサポートしながら、複合コンポーネントに対するサポートも採用します。

## JSFページのブックマーキング

JSF 2.0では、GETリクエストの処理とパラメータ・マッピングに対するサポートが提供されているため、開発者は、ブックマークから参照されたときに動的な状態を再作成する方法を認識しているJSFページを構築できます。

また、Oracle ADFコントローラ内のアンバウンド・タスク・フローでも、ブックマーキングのサポートが提供されています。こちらは、JavaServer Faces 2.0でのブックマーキングの実装方法と比較すると、ページ・ソースでパラメータ・マッピングを定義したり、GETリクエストに対応した特定のUIコンポーネントでナビゲーションを実行したりする必要がありません。JavaServer Faces 2.0を採用しても、ADFコントローラは引き続きブックマーキングをサポートします。

## 暗黙ナビゲーション

特定のアクション名を持つビューIDが存在する場合は、ナビゲーション・ケースを定義する必要のない、JavaServer Faces 2.0の暗黙ナビゲーションは、ADF Facesでサポートされる予定ですが、Oracle JDeveloper 11g 11.1.2では使用できない可能性があります。

ADFコントローラの開発の背後にある動機付け要因の1つに、メソッド・コール・アクティビティやルーターの決定のような目に見えないターゲットにナビゲーションをルーティングする機能があります。JavaServer Faces 2.0は引き続き、ビュー間のナビゲーションのみをサポートします。つまり、プリエンティブ・ナビゲーションを使用して、メソッドのような目に見えないアクティビティにナビゲートすることはできません。

## アノテーション

JSF 2.0は、マネージド・ビーンをfaces-config.xmlファイル内で構成する代わりに、マネージド・ビーンを定義するためのアノテーションをサポートします。ADFコントローラを利用した場合、マネージド・ビーンをバウンド・タスク・フローで使用するよう構成できますが、それには構成ファイルでの登録が必要です。この点を考慮すると、JavaServer Faces 2.0プロジェクト内でのナビゲーションのためにADFコントローラを使用する場合は、構成ファイル内にビーン定義が引き続き必要になります。

## エラー処理

JSF 2.0では、リクエスト・ライフ・サイクル中に発生した予期しない例外を集中処理するための新しいExceptionHandler APIが導入されています。ADF Facesにも、引き続きサポートされる同様のサービスが用意されています。ADF Facesの例外ハンドラによって処理されない例外は、JSF 2.0の例外ハンドラに委任され処理されます。

## 検証

JSF 2.0では、ビーン検証が使用可能なときに空のフィールドを検証できるようにするためのjavax.faces.VALIDATE\_EMPTY\_FIELDSコンテキスト・パラメータが導入されています。この検証は、JavaServer Faces 2.0の要件に従って、ADF Facesのために実装および確認されています。

JSF 2.0での検証の別のおもな機能として、f:validateBeanタグを使用したJSR-303 (Bean Validation) との統合があります。f:validateBeanタグを使用したビーン検証のADF Facesとの統合の影響は現在確認中であり、間もなくサポート対象になる予定です。

## ネーミング・コンテナのセパレータ

JSFでのネーミング・コンテナは、ビューの範囲内でのネーミングの競合を防止するために子コンポーネントに対する一意のネーミングを保証する名前空間に似ています。JSF 2.0では、ネーミング・コンテナIDをコンポーネントIDから区別するためにデフォルトで使用されるコロン文字を、javax.faces.SEPARATOR\_CHARコンテキスト・パラメータを使用して構成できます。ADF Facesは、JSF 2.0サポートの以降のリリースで、この要件を解決する予定です。

## プロジェクト・ステージ

JSF 2.0のプロジェクト・ステージ機能はADF Facesでサポートされます。開発者はプロジェクト・ステージを使用して、特定の実行時環境のためのADF Facesコンテキスト設定を定義します。たとえば、開発者は開発中に、テストするアプリケーションのデバッグやアサーションを有効にしたり、コンテンツ圧縮を無効にしたりすることができます。ただし、本番モードではこれらの設定は推奨されず、警告が出力されます。

## ADF Faces JSF 2.0の採用 : Q & A

以下では、開発者が既存のADF FacesアプリケーションをJavaServer Faces 1.2からJavaServer Faces 2.0に移行する場合に抱く可能性のある開発に関する質問に答えています。

### Oracle JDeveloperリリース固有の質問

**JavaServer Faces 2.0をサポートしているのはOracle JDeveloperのどのリリースですか。**

Oracle JDeveloper 11g Release 2 (11.1.2) が、JSF 2.0をサポートする最初のJDeveloperリリースです。

**JDeveloper 11g Release 2でJSF 1.2ベースのアプリケーションを開発するにはどうすればよいですか。**

Oracle JDeveloperの新しいリリースを使用してJSF 1.2ベースのアプリケーションを保守またはさらに開発しようとするアプリケーション開発者は、現在のプロジェクト依存性をJSF 2.0に移行する必要があります。最新バージョンのOracle JDeveloper 11gでは、JavaServer Faces 1.2がサポートされなくなっています。JSF 2.0に移行したくない開発者は、Oracle JDeveloper 11.1.1.xを使用し続ける必要があります。

**既存のADF FacesアプリケーションをJSF 2.0に移行するにはどうすればよいですか。**

既存のJSF 1.2ベースのアプリケーションが最初に開かれると、JDeveloperはそのアプリケーションをJSF 2.0に自動的に移行します。この時点で、JDeveloperには、ユーザーに移行しないようにするオプションを提供するアラートが表示されます。移行を止めた開発者は、Oracle JDeveloper 11g Release 2でそのアプリケーションを開くことができないため、Oracle JDeveloper 11g Release 1を使用し続ける必要があります。移行のダイアログを受け入れると、プロジェクトのライブラリ依存性がJSF 2.0に変更されます。アプリケーション・コードに変更は適用されません。

ADF FacesアプリケーションがJavaServer Faces 1.0とADF Faces 10.1.3を使用している場合は、移行パスがまず11gへのアップグレードになります。これは、Oracle JDeveloper 11g Release 1を使用して実行されます。

## ADF Faces固有の質問

### JavaServer Faces 2.0でJSF 2.0 RIコンポーネントではなく、ADF Facesを使用する利点は何ですか。

Oracle ADF Facesは、単なるJSFコンポーネント・セットを超えたものになっています。それは、JSF標準APIの上に構築されたビュー・レイヤー開発フレームワークです。ADF Facesは、データ可視化コンポーネント、グラフィカルなマップやリージョンなどの、JSF 2.0のリファレンス実装には存在しないコンポーネントを含む150を超えるコンポーネントを提供します。

追加のコンポーネントとともに、ADF Facesでは、ドラッグ・アンド・ドロップ、軽量ダイアログ処理、アクティブ・データ・ストリーミング、JSR-227ベースのADFバインディング・フレームワークとの統合などのサービスのほか、JSF標準では使用できない完全なJavaScriptコンポーネント・クライアント・アーキテクチャが提供されます。

### ADF FacesアプリケーションはJSF 2.0と統合されますか。

ADF Facesアプリケーションは、JSF 2.0に移行されたか、またはOracle JDeveloper 11g Release 2を使用して新しく開発されたかを問わずJavaServer Faces 2.0と統合されるため、新しい標準機能を使用できます。下位互換性を維持するとともに、JSF 2.0では使用できない機能をさらにサポートするために、既存の高度なADF Faces機能は引き続きサポートされます。

### 移行されたアプリケーションはFaceletsを使用することを期待されますか。

既存のADF Facesアプリケーションは、VDLをFaceletsに変更する必要はなく、引き続きJSPドキュメントを使用できます。

”この仕様への準拠を主張するJavaServer Faces実装はすべて、JavaServer Pagesの完全な実装を含め、この実装をすべてのJSFアプリケーションのランタイムに公開する必要があります。ただし、JSFアプリケーションは、ビュー宣言言語 (VDL) としてJSPを使用する必要はありません”

- 『JavaServer™ Faces Specification 2.0』の第9章

ただし、JSF 2.0標準に追加された新機能には、ビュー宣言言語としてFaceletsが必要です。現在JSFでJSPドキュメントが使用されている場合、オラクルでは、できるだけ早くFaceletsに移行することを推奨しています。

Faceletsは、JSFを念頭において最初から設計されたJSPの後継です。バージョン2で導入された新機能は、Faceletsを使用しているページ作成者にのみ公開されます。JSPは下位互換性のために保持されています。

- 『JavaServer™ Faces Specification 2.0』の第10章

JavaServer Faces 1.2でFaceletsを使用している顧客は、JavaServer Faces 2.0で標準化されたFaceletsのバージョンに、以前のオープンソース版への完全な下位互換性がないことに注意してください。この点はJSF 2.0仕様にも明示されています。

VDLとしてJSPを使用している既存のADF Facesアプリケーションは、Faceletsに移行する必要はありません。ただし、Faceletsの優位性から次のような利点が得られます。

- JSFとのより明確な統合 - Faceletsは、JSFのユースケースを対象とするように最初から設計されました。それに対して、JSFのサポートはどちらかと言うと、既存のJSPアーキテクチャに後から追加されました。そのため、Faceletsによって、JSFとのより明確な統合が提供されます。
- コンパイルは必要なし - FaceletsビューはFaceletsエンジンによって直接解析されるため、ページ定義をJavaに変換する必要はなくなります。

- ページ・テンプレートの改善 - Faceletsによって、ページ・テンプレート化の統合が可能になります。
- エラー・レポートの改善 - EL評価の例外などのランタイム・エラーは、JSPエンジンによって必ずしも問題が発生しているドキュメント内の場所を示す明確なガイダンスが示されずには限らないため、追跡が困難な場合があります。Faceletsでは、ページ上の失敗したメタデータ定義の行番号を含むより詳細な情報が提供されるため、デバッグが容易になります。
- パフォーマンスの向上 - Faceletsは、実行時にJavaコードにコンパイルする必要がないため、パフォーマンスが向上します。

#### JSPからFaceletsドキュメントに移行しない理由がありますか。

FaceletsとJSPドキュメントは、実行時にJSFビューの定義を異なった方法で処理します。JSPドキュメントは、JSFページの定義を、サーバー側のJSPランタイム・エンジンによって完全に解析および処理されたXMLメタデータとして処理します。FaceletsはJSFページを、サーバー側で部分的に処理され、残りがブラウザによって処理されたXHTMLドキュメントとして扱います。

JSPドキュメントをJSFページのVDLとして使用し続けるおもな理由は、JSFのエキスパート・グループが、クライアントによって処理されることを目的としたコンテンツのみがクライアントに渡されるようにFaceletsをXML認識型にするよう改善するのを待っているからです。

Faceletsへの移行を待つもう1つの理由として、あるプロジェクトが、まだFacelets対応機能を備えていないライブラリに依存している場合があります。

**注：**VDLとしてのJSPからFaceletsへの自動移行は、Oracle JDeveloper 11g Release 2リリースの範囲外です。新しいJDeveloperリリースで開かれたときにJSF 2.0に移行されたアプリケーションは、引き続きJSPドキュメントを使用します。

#### 将来Faceletsに移行したいと考えている場合、現在は何を考慮しておく必要がありますか。

将来のFaceletsへの移行を準備するには、次の点に注意してください。

- JSPドキュメント内のCDATAブロックは避けてください - Oracle JDeveloper 11gの初期バージョンには、JavaScriptやCSSのソースを保持または参照するためのaf:resourceタグは存在しませんでした。その時点のベスト・プラクティスは、af:document metaContainerファセット内でCDATAブロックを使用してこれらのリソースを含めることでした。JSF 2.0に移行するときは、JavaScriptやCSSのリソースへのすべての参照をaf:resourceタグの使用に変更することを推奨します。

- JSFドキュメントではスクリプトレットを使用しないでください - まれなユースケースですが、開発者は依然として、JSFページ上で参照されるJSTL内でスクリプトレットを使用しています。Faceletsはスクリプトレットをサポートしていないため、ADF Facesページ内のスクリプトレットをすべて削除する必要があります。
- JSPドキュメント内のコメントに注意してください - JSPドキュメント内のコメントは、削除されない限りFaceletsページに配信されます。Faceletsには、コメントがクライアントに送信されないようにするためのコンテキスト・パラメータがあります。
- JSPXドキュメントを使用してください - Faceletsには有効なXMLドキュメントが必要ですが、これはJSPXドキュメントを使用して準備するのが最適です。

**オラクルはADF FacesでのJSPXドキュメントの使用をいつまでサポートする予定ですか。**

オラクルは、開発者がJSF 2.0の新機能を使用したくなるものと期待しています。JSF 2.0の新機能の多くには、VDLとしてFaceletsが必要です。JSPからFaceletsへの許容可能な移行パスが用意されるまで、オラクルがADF FacesでのJSP VDLのサポートを打ち切る予定はありません。

ただし、開発者が既存および新規のADF Facesアプリケーションのために、できるだけ早くFaceletsを採用することを推奨します。

**ADF FacesはFaceletsとJSPXに対して同じ機能やコンポーネントを提供しますか。**

Oracle JDeveloper 11g Release 2でのJSF 2.0上のADF Facesの初期リリースではそうなります。ただし、JSF 2.0の一部の機能はFaceletsに対してのみ使用可能であり、これが間違いなく将来のADF Facesの新しい機能に影響を与えます。

**ADF Facesのページ・テンプレートはFaceletsで動作しますか。**

はい。ADF Facesのページ・テンプレートはFaceletsで使用できます。

**ADF Facesの宣言的なコンポーネントはFaceletsで動作しますか。**

宣言的なコンポーネントに対する計画は、開発者がFaceletsでも使用できるようにすることです。このため、宣言的なコンポーネントをJDeveloper 11g Release 2以降で開いた後、再デプロイする必要があります。ただし、オラクルでは、既存の宣言的なコンポーネントのFaceletsへの移行を提供していません。

**宣言的なコンポーネントを引き続き使用する理由と時期を教えてください。**

JavaServer Faces 2.0の新しい複合コンポーネント機能は、開発者に、既存のJSFコンポーネントからカスタム・コンポーネントを構築するためのオプションを提供します。ただし、この機能はVDLとしてJSPではなく、Faceletsを使用した場合にのみ動作します。

ADF Facesの宣言的なコンポーネントはJSPドキュメントで動作するほか、JDeveloper 11g Release 2で作成された新しい宣言的なコンポーネントの場合はFaceletsでも動作します。

ADF Facesの宣言的なコンポーネントのためのユースケースは次のとおりです。

- ページ上にコンポーネントの複数のインスタンスが存在する可能性があり、かつ開発者が各コンポーネント・ビーンを分離できる必要がある場合の、コンポーネントによるバックギング・ビーンの使用。
- 開発者が引き続きJSPを使用する必要がある場合。

**ADF Facesコンポーネントを使用してJSF 2.0の複合コンポーネントを構築できますか。**

はい、できます。

**他のサード・パーティ製のJSFフレームワーク内でADF Facesコンポーネントを使用できますか。**

ADF Facesはドキュメントとフォームを所有する必要があります。つまり、`af:document`タグと`af:form`タグを使用する必要があります。そのため、任意のページ内へのADF Facesコンポーネントの配置は、まだサポートされていません。他のコンポーネント・フレームワークにも同様の制限がある可能性があるため、ADF Facesの`af:document`タグと`af:form`タグが使用されている場合でも、この2つのコンポーネント・セットの組合せが動作しないか、またはサポートされない可能性があることに注意してください。

**JSF 2.0のAPIコールとADF FacesのAPIコールを混在させることはできますか。**

はい、できます。ADF FacesとJSF 2.0の両方に同じ機能を見つけた開発者は、自由に好きな方を選択できます。ただし、ADF FacesのJSF 2.0サポートの初期リリースでは、ADF FacesのAPIを使用することを推奨します。

**ADF Facesで`jsf.ajax.request()`の呼出しを使用できますか。**

はい、できます。ただし、JSF 2.0の`f:ajax`タグによる宣言的なアプローチ、または既存のADF Facesの部分送信および部分トリガー実装を使用することを推奨します。

**ADF Facesの部分トリガーはJSF 2.0のAjaxリクエストとどのように統合されるのですか。**

発行側のコンポーネントがADF Facesコンポーネントである場合は、部分トリガーが評価され、参照されたコンポーネントが部分ターゲットのリストに追加されます。AjaxリクエストがADF Faces以外のJSFコンポーネントによって発行された場合、部分トリガーは解決されません。サーバー側のイベント・ハンドラ内で`RequestContext.partialUpdateNotify()`を呼び出すことによって、部分トリガーを手動で強制的に解決できます。

注：`RequestContext`は、`AdfFacesContext`クラスによってラップされた`Trinidad`クラスです。そのため、`AdfFacesContext.partialUpdateNotify()`を呼び出すこともできます。

**ADF FacesはJSF 2.0にある新しいエラー処理APIを利用していますか。**

ADF Facesアプリケーションに対して例外ハンドラが明示的に構成されていない場合は、JSFのデフォルトの例外処理メカニズムが使用されます。ADF Facesのために構成された例外ハンドラの場合でも、構成されたハンドラを超えて伝播される例外はすべて、可能であれば、JSF 2.0のハンドラに渡されて処理されます。ただし、ADF Facesは引き続き、既存のoracle.adf.view.context.ExceptionHandler実装をサポートします。

## ADFバインディング固有の質問

**JSF 2.0を採用するとき、ADFのバインドされたアプリケーションを変更する必要がありますか。**

ADF FacesをJSF 1.2からJavaServer Faces 2.0に移行するとき、Oracle ADFバインディング・レイヤーは影響を受けません。Oracle JDeveloper 11g Release 2で既存のADFアプリケーションを開くと、ADFバインディング・ライブラリは、最新バージョンのFacesコントロール・バインド・クラスを含むように自動的に更新されます。

## ADFコントローラ固有の質問

**ADFコントローラでアノテーションを含むマネージド・ビーンを定義できますか。**

現在、ADFコントローラはアノテーション付きマネージド・ビーンをサポートしていません。すべてのマネージド・ビーンの使用をタスク・フローのメタデータ・ファイルまたはadfc-config.xmlファイルで定義する必要があります。

**JSF 2.0のビュー・スコープまたはフラッシュ・スコープ内でマネージド・ビーンを定義できますか。**

ADFコントローラは（したがってタスク・フローも）現在、JavaServer Faces 2.0のフラッシュ・スコープをサポートしていません。マネージド・ビーンがviewScope内で構成された場合は、ADFコントローラのビュー・スコープが使用され、これによってJSF 2.0内の同じ名前を持つスコープ・オブジェクトが拡張されます。

**ADFコントローラはJSF 2.0のプリエンティブ・ナビゲーションをサポートしますか。**

ADFコントローラのNavigationHandlerは、ConfigurableNavigationHandlerを実装します。つまり、これをプリエンティブ・ナビゲーションに使用できます。ただし、一致するナビゲーション・ルールに、メソッド・コール・アクティビティなどの目に見えないアクティビティが含まれている場合は、ナビゲーションの結果をプリエンティブに決定できないため失敗します。

## 結論

Oracle JDeveloper 11g Release 2（バージョン11.1.2）で、ADF FacesはJavaServer Faces 2.0を利用するようになりました。つまり、開発者は現在、JSF 2.0の新機能を開発プロジェクトに統合を選択できます。

ADF FacesのJavaServer Faces 2.0との統合は、Oracle JDeveloper 11g Release 2のリリースによって開始されたばかりの魅力的なプロセスであり、以降のリリースでも引き続き大幅に拡張されていく予定です。ADF Facesの目標は、JSF 2.0に移行しながら、開発者に比類ないレベルの機能と生産性を引き続き提供することです。ADF Facesフレームワークの観点から見ると、JSF 2.0の採用におけるほとんどの変更は内部的であるため、開発者には透過的であるはずですが、



JavaServer Faces 2.0の概要および  
Oracle ADF FacesとOracle  
JDeveloper 11gでの導入計画  
2010年12月

著者：Frank Nimphius  
共著者：Andy Schwartz、  
Duncan Mills、  
Shaun O'Brien

Oracle Corporation  
World Headquarters  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U.S.A.

海外からのお問い合わせ窓口：  
電話：+1.650.506.7000  
ファクシミリ：+1.650.506.7200  
www.oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. 本文書は情報提供のみを目的として提供されており、ここに記載される内容は予告なく変更されることがあります。本文書は、その内容に誤りがないことを保証するものではなく、また、口頭による明示的保証や法律による黙示的保証を含め、商品性ないし特定目的適合性に関する黙示的保証および条件などのいかなる保証および条件も提供するものではありません。オラクル社は本文書に関するいかなる法的責任も明確に否認し、本文書によって直接的または間接的に確立される契約義務はないものとします。本文書はオラクル社の書面による許可を前もって得ることなく、いかなる目的のためにも、電子または印刷を含むいかなる形式や手段によっても再作成または送信することはできません。

Oracleは米国Oracle Corporationおよびその子会社、関連会社の登録商標です。その他の名称はそれぞれの会社の商標です。  
0109