Oracleホワイト・ペーパー
2010年6月

タスク・フロー設計の基礎

ORACLE

概要

コントローラのタスク・フローは、任意のOracle Application Development Framework（Oracle ADF）アプリケーションの主要な要素の1つです。JSFまたはStrutsの経験がある場合は、タスク・フローをこれらのコントローラ・フレームワーク内のページ・フローと同じと見なすことがありますが、タスク・フローはOracle ADFコントローラの機能の大幅な活用に基づいています。本書では、コントローラの機能を最大限に活用するタスク・フローを設計するための考慮事項とベスト・プラクティスのいくつかについて説明します。また本書は、読者がすでにOracle ADFコントローラで使用されている主要な概念やコンポーネントに精通していることを前提にしています。そうでない場合は、先に進む前に『Oracle Fusion Middleware Oracle Application Development Framework のための Fusion開発者ガイド』（http://www.oracle.com/technology/documentation/jdev.html）にあるこの件に関する内容をご覧ください。

どのような種類のタスク・フローか

開発者がOracle ADFコントローラで作業し始める際に、最初に尋ねることが多いのは、バウンド・タスク・フローとアンバウンド・タスク・フローの区別に関する質問です。Oracle ADFコントローラを使用するプロジェクトではすべて、アンバウンド・タスク・フローの定義（adfc-config.xml）がデフォルトで作成されます。機能の観点から見ると、ここで考慮に入れるべき、アンバウンド・タスク・フローに関連した4つの重要な考慮事項があります。

1. 1つのアプリケーション内に存在するアンバウンド・タスク・フローは1つだけです。そのアプリケーション内でアンバウンド・タスク・フローの複数の定義ファイルを使用した場合でも、実行時に1つのフラットな名前空間に結合され、起動時にフロー全体がメモリにロードされます。
2. アンバウンド・フロー内のビュー・アクティビティはすべて、ユーザーがURLでアクセスできます。
3. アンバウンド・タスク・フローは、ページ全体として実装されるビュー・アクティビティに制限されます。
4. ブックマーキングはアンバウンド・タスク・フロー内のビューでのみサポートされ、バウンド・タスク・フローでは使用できません。

これらの考慮事項に基づくと、実際に、アンバウンド・タスク・フローはユーザーのアプリケーション（たとえば、ホームページや、電子メールから深くリンクされているページなど）へのエントリ・ポイントを定義するためにのみ使用する必要があることがわかります。断片ベースのページ・アセンブリを最大限に活用する標準的なアプリケーション内では、これは通常、アンバウンド・タスク・フロー内のビュー・アクティビティが実際に1つしか存在しないことを示すことに他なりません。

バウンド・タスク・フローは、この他のすべての状況に使用する必要があります。ほとんどの状況で、これらのバウンド・タスク・フローはアンバウンド・タスク・フロー上のコア・ページ内でホストされるため、ページ全体ではなくJSF断片で構成されます。

タスク・フロー設計の一般的な原則

状態の管理

他の任意のJava EEアプリケーションと同様に、開発者は、作成するすべてのオブジェクトの処理に細心の注意を払う必要があります。一般に、セッション・スコープのマネージド・ビーンはadfc-configのレベル（アンバウンド・タスク・フロー）でのみ定義する必要があり、真にセッション全体にわたって存在する必要のあるオブジェクトにのみ使用する必要があります。リクエストより長いスコープまたはviewScopeが必要なオブジェクトは一般に、セッション・スコープではなくpageFlowScopeを使用する必要があります。これにより、状態の分離や、アプリケーションに対するよりコンポーネント指向の開発アプローチが可能になるためです。また、フロー固有の情報をpageFlowScopeに格納すると、特定のタスク・フロー・インスタンスに関連するpageFlowScopeが終了時に自動的にクリーンアップされるため、アプリケーションの長期にわたる潜在的なメモリ要件も制限されます。

一時的な状態を通る必要のあるページ断片の場合は、この状態を処理する場所を決定するときに、リクエストとviewScopeのライフタイムの違いを考慮してください。断片ベースのタスク・フローのコンテキストでは、それらの断片を含むサブフローのコンテキスト内のより予測可能な動作に対して、リクエスト・スコープではなくviewScopeを使用していることに気付くことがあります。

バッキング・ビーン・スコープは、同じUI断片内で複数回利用される（たとえば、表やその他の繰り返しコンポーネント内で消滅する）宣言的なコンポーネントに使用する必要があります。これにより、そのコンポーネントの各インスタンスが、それぞれ独立した状態を維持することが保証されます。バッキング・ビーン・スコープはまた、同様に同じページ上で複数回利用される可能性のあるリージョンを管理しているビーンにも使用できます。

各スコープについて詳しくは、付録Aにあるスコープ・ダイアグラムを参照してください。

例外処理

タスク・フローには常に、例外ハンドラとしてマークされたアクティビティが含まれている必要があります。このアクティビティはビュー・アクティビティである必要はなく、以降のアクティビティに対する制御フロー・ルールを備えたメソッドまたはルーター（推奨されるアプローチ）であってもかまいません。タスク・フロー・コール・アクティビティと相互接続されたタスク・フローを使用する場合、例外ハンドラは必ずしも必要ではありません。その場合は、例外がすべて親にバブルアップされます。ただし、これは必然的にタスク・フローでの情報漏えいにつながるため、明示的なハンドラを指定することを常に推奨します。ページ上のリージョン内に埋め込まれたタスク・フローの場合は、例外処理アクティビティを指定する必要があります。

例外処理に対する一貫性のあるアプローチを保証するために、次のことが推奨されます。

1. 例外処理アクティビティを、タスク・フロー・テンプレート内にルーター・アクティビティの形式で作成します（下を参照）。作成するバウンド・タスク・フローからこのテンプレートを参照します。
2. タスク・フローの例外ハンドラ（ルーター）には、"未処理"として定義されたデフォルトのナビゲーション・ケースが存在する必要があります。未処理の例外とは、テンプレート内の例外ハンドラによって処理されていない（そのため、一般化できないバウンド・タスク・フローに特有の）例外のことです。

3 "未処理"というワイルドカードのナビゲーション・ケースは、タスク・フロー・テンプレートを継承するバウンド・タスク・フローで作成されます。この"未処理"のフローはその後、例外メッセージのためにコントローラ・コンテキストにアクセスできるルーター・アクティビティに移動されるため、タスク・フローに固有の操作を実行できるようになります。

タスク・フロー・テンプレート

タスク・フロー・テンプレートを使用すると、開発者は、以降に作成されるフローに継承またはコピーされる共通の機能を定義できます。継承のアプローチを使用している場合、このテンプレートは、共通のマネージド・ビーン、上述の例外処理フロー、トランザクション動作などの共通のタスク・フロー設定を定義するために使用する必要があります。

継承されたテンプレート・タスク・フロー内で定義されたタスク・フロー・アクティビティは、依存するフローの設計時のビューには表示されません（ただし、名前空間は実行時に結合されます）。そのため、テンプレート内で、作成した明示的な制御フロー・ケースとの関係を持つアクティビティを定義するべきではありません。定義しても、この関係を、消費側のタスク・フロー・ダイアグラムにあるこれらのアクティビティに伝えることができません。

テンプレート・タスク・フローを参照するのではなく、コピーすることを選択する場合は、このテンプレートへの以降の変更が新しく作成されたフローに反映されないことに注意してください。ただし、テンプレートのコピーのアプローチが適切な場合もあります。たとえば、次の場合があります。

Person、Order、Storeなどのエンティティ上で動作するCRUDタスク・フローは、そのエンティティを作成、更新、削除、および表示する必要があります。このために、デフォルト・アクティビティとしてのルーター、編集のためのビュー、読取り専用表示のためのビューや、エンティティ・オブジェクトを問い合わせ、それを削除するためのメソッド・コール・アクティビティなどの同様のプロセス手順を使用します。タスク・フローの分離やトランザクション・レベルに関連したプロパティ設定も含むことのできるこのようなブループリントをタスク・フロー・テンプレートで定義し、後でフローの実装時にコピー・ソースとして使用できます。このような場合、タスク・フロー・テンプレートの継承は必要ありません。このアプローチを使用する場合は、MDSのカスタマイズに使用されるタスク・フローのメタデータID値の一意性を保証する必要があることに注意してください。

また、名前の衝突や、テンプレート定義の不注意による上書きのリスクを最小限に抑えるために、タスク・フロー・テンプレートによって定義されたすべてのアクティビティ、パラメータ、またはマネージド・ビーンの名前付けスキームを標準化することも必要です。

制御フロー・ルール

フロー内で明示的な制御フロー・ルールとワイルドカード制御フロー・ルールのどちらを使用するかに関して、絶対的な基準はありません。ただし、タスク・フロー・ダイアグラムを明確で、わかりやすい状態に保つのに役立つように、制御フロー・ルールのワイルドカード機能を使用することを推奨します。たとえば、タスク・フロー内の"キャンセル"や"リターン"などの一般に使用されるアクションは複数のアクティビティからアクセスされる可能性があるため、1つのワイルドカード・ルールでダイアグラムをはるかに明確な状態に保つことができます。

ワイルドカード・ルールでは、特定の結果がワイルドカード・ルールにマッピングされているという事実を開発者がすばやく識別できるように、一貫性のあるネーミング規則を使用するようにしてください。ここでは、ルールの名前に"global"などの接頭辞を使用すると有効な場合があります。

フローの注記、注釈、その他のドキュメント

タスク・フローを文書化することによって、付加価値が与えられたダイアグラムの注釈や添付ファイルによって提供される機能を利用します。

より複雑なドキュメント要件の場合は、タスク・フロー内に追加のHTMLドキュメントを埋め込むことを検討してください。各タスク・フローを記述するためのいくつかの標準テンプレートや、この追加のドキュメントのための標準化された場所を確立することもできます。

コンポーネントとしてのタスク・フロー

概要

タスク・フローを、開発者の間でコンポーネントとして渡される再利用のメカニズムとして使用している場合は、タスク・フローとその消費側のページ（またはフロー）の間のインタフェースに特別な注意を払う必要があります。バウンド・タスク・フローを、入力のためのパラメータと出力のための戻り値を備えた、かなり単純なコンポーネントであると考える傾向がありますが、作業単位を記述したアトミック・サービスと見なす方が適切です。経験によると、明らかなアトミック・サービスでさえ、一般に予測されるよりはるかに複雑です。開発者は、タスク・フローが外部情報をどのようにして"漏えい"させるか、またはそれ以外の方法で外部情報に依存する可能性があるかを理解している必要があります。

次に、以下のインタフェース・ポイントについてさらに詳細に説明します。

- パラメータ
- 戻り値
- ナビゲーションの副作用
- コンテキスト・イベント
- データ・コントロール・スコープ
- セッション・スコープ変数
- リクエスト・スコープ変数
- セキュリティ・ロール
- ファイル・アップロード
- その他のコンテキスト情報
- リソース・バンドル

パラメータ

パラメータはタスク・フローのAPIで重要な役割を果たし、フローとそのコール元の間の疎結合を保証します。パラメータを定義する場合は、次のルールに従う必要があります。

1. 設計時および実行時環境で特定のパラメータがオプションか必須かを確実に識別できるように、任意のパラメータの必須の属性を正しく使用します。
2. パラメータを定義する場合は、パラメータの"値"（宛先）を明示的にマッピングしなかった場合に設定される、pageFlowScope内の暗黙のパラメータ記憶域に依存しないでください。一致するpageFlowScopeマネージド・ビーンをタスク・フロー定義で常に明示的に定義し、これをパラメータの値パラメータとしてマッピングしてください。この宛先のビーン定義には、渡されるパラメータの型を含めることができます。これにより、実行時に型がチェックされ、パラメータの型が間違っている場合は例外が発生することが保証されます。また、タスク・フロー内にマネージド・ビーンの定義を含めると、設計時に、そのフロー内の以降のアクティビティでのそのデータの使用が正しくサポートおよび監査されることも保証されます。

3 入力パラメータへのアクセスが必要なフロー内の式をマッピングする場合は、これらの式を暗黙のパラメータ名変数（式エディタで緑色の立方体のアイコンが付いています）ではなく、これらの受信者のマネージド・ビーン（式エディタでビーンのアイコンが付いています）にマッピングするようにしてください。

戻り値

戻り値パラメータは、フローへの入力パラメータと同様にAPIの重要な要素ですが、その結果は型チェックされません。そのためここでも、実行時の型チェックを保証するために、呼び出し元のフロー内で受信側のビーンを明示的に定義する必要があります。

ナビゲーションの副作用

埋め込まれたタスク・フロー内のナビゲーションの操作によって、親のタスク・フローに次の2つの異なる方法でアクションが生成される可能性があります。

1 ペアレント・アクション・アクティビティは、バウンド・タスク・フローの開発者に親のアクションをトリガーするための方法を提供します。そのため、フローによって報告される可能性のある親の任意の結果は、パラメータや戻り値と同程度にこのフローのAPIの一部になります。このようなアクティビティの使用を少数の明確に定義された制御フロー・ルールに制限するとともに、タスク・フローのコンシューマに対して正確なイベント・ポイントを文書化する必要があります。

2 リージョン・ナビゲーション・リスナーは、ページ内に埋め込まれた断片ベースのバウンド・タスク・フローの場合に固有のものです。この場合、このフローが含まれているページは、regionNavigationListenerを介してそのフロー内で実行されるナビゲーションにアクセスします。この場合は、埋め込まれたフローの観点から見て、その接続は受動的です。そのため、リージョン・リスナーは、タスク・フローから一部の情報を漏えいさせるための方法を提供するにもかかわらず、実際には正式なタスク・フローAPIの一部ではありません。再利用可能なタスク・フローの作成者は、必要に応じて、親へ通知するためのより明示的なメカニズム（たとえば、コンテキスト・イベントやペアレント・アクション）を検討する必要があります。

これらの副作用の可能性があるため、ペアレント・アクション・アクティビティ・タイプの最善の使用を、バウンド・タスク・フローがその親と緊密に結合されている場合とすることを推奨します。そのため、ペアレント・アクティビティを使用して、1つの再利用可能なコンポーネント内に束ねられているネストされたタスク・フロー間のナビゲーションを生成することは妥当です。ただし、このようなアプローチをコンポーネントのトップ・レベルのタスク・フローで使用するのは、消費側のタスク・フローの構造に関する前提条件が必要になるため良い考えではありません。このような場合は、コンテキスト・イベントによって提供される、より疎な結合を使用する必要があります。

コンテキスト・イベント

コンテキスト・イベントは、タスク・フロー間の疎結合のパブリッシュおよびサブスクライブ・イベント処理モデルを作成するための、Oracle ADFバインディング・レイヤー上のメカニズムを提供します。イベントは、埋め込まれたタスク・フローからの出力と、そのタスク・フローへのサブスクライブの両方が可能であり、データ・ペイロードを運ぶことができます。多くの場合は、システム内のデータ表示を同期するために使用されます。

コンテキスト・イベントはタスク・フローの設計者にきわめて強力なツールを提供するため、インバウンドとアウトバウンドの両方の場合について完全に文書化される必要があります。

コンテキスト・イベントを使用する設計を検討する場合は、次の点に注意してください。

- アプリケーションの他の部分でADFmが使用されていない場合でも、コンテキスト・イベントは、Oracle ADFバインディングのメカニズムを使用して登録とイベント伝播の両方を処理します。
- コンテキスト・イベントは、ページ上のリージョン内に断片ベースのタスク・フローが埋め込まれている場合にのみ適用されます。
- 共有されたデータ・コントロール・スコープを継承するタスク・フローを実行する場合は、データ同期のためにコンテキスト・イベントを使用する必要はありません。データが含まれているリージョンをPPRでリフレッシュして最新の行の基準を取得し、表示されているデータをリフレッシュするだけで十分です。
- イベントとともに渡されるペイロードが、1つの単純なスカラー型である必要はありません。受信側のメソッドはこの点を利用し、必要なAPIを効率的に実施するために必要となる数の型を持つ引数を提供する必要があります。

データ・コントロール・スコープ

Oracle ADFバインディングおよびデータ・コントロールを利用する任意のタスク・フローにおける重要な考慮事項に、データ・コントロール・スコープ（タスク・フロー・エディタの「Overview」→「Behavior」タブを使用して設定）があります。この値はデフォルトではtrueに設定されるため、開発者が、データ・コントロール・コンテキストから何らかの暗黙の状態を継承すると予測できることを示します。この情報には、次のものが含まれます。

- コレクション内の行の基準などの、既存のデータ構造および属性の予測（たとえば、ビュー・オブジェクトの結果セット）。
- 共有されたトランザクション・コンテキスト。この場合、タスク・フロー内のコミットやロールバックに、親のデータ・コントロール・コンテキストをコミットする（またはその逆）という副作用が伴います。

データ・コントロール・スコープが共有されていない場合は、これによってタスク・フロー内の機能のより厳密なカプセル化が維持されますが、それがデータベース接続やキャッシュのメモリ使用量の増加を犠牲にして得られることに注意してください（Oracle ADF BCが使用されていると仮定した場合）。

タスク・フローのドキュメントでは、データ・コントロール・スコープが共有されているかどうか、共有されている場合はどのような依存関係が存在するか（該当する場合）について特に明確に記述してください。

セッション・スコープ変数

一般的な原則として、アプリケーション内のセッション・スコープ属性は慎重に管理する必要があります。これらの属性は文書化しにくく、アプリケーション内に多くの隠された依存関係を生成する可能性があります。任意のOracle ADFアプリケーションでの一般的な規則として、セッション・スコープ変数は例外としてのみ作成するようにしてください。

セッション・スコープ（およびアプリケーション・スコープ）変数はもちろん、アプリケーション内のすべてのタスク・フローから常に使用できます。これらを読取り専用のコンテキスト以外では決して使用しないようにすること、また、その場合でも、タスク・フローに非公開コピーが残るようにデータをpageflowスコープ変数にマッピングすることを推奨します。

タスク・フロー内からセッション・スコープ変数にアクセスする場合は、複数の同時実行タスク・フローがその状態に書込みを行う可能性があることに注意してください。そのため、特定のフローから見て、その値は時間の経過とともに変化する可能性があります。

セッション・スコープおよびアプリケーション・スコープ属性は文書化しにくく、どこで使用され、いつクリーンアップする必要があるかを識別することは困難です。さらに、セッションおよびアプリケーション・スコープ変数は親アプリケーションに属しているため、タスク・フロー内に実行環境への依存関係が生成され、再利用性が低下します。

外側のフローにデータを非同期的に渡す必要がある場合は、セッション変数ではなく、コンテキスト・イベントをメカニズムとして使用してください。

リクエスト・スコープ変数

呼び出されたタスク・フロー内のセッション・スコープ・データのアクセス性は自明のことです。ただし、requestScopeがフローの作成とわずかにオーバーラップし、パラメータのメカニズムを通さずに情報をタスク・フローに漏えいさせることができる可能性があることにも注意してください。パラメータ属性の設定で説明した特殊なケースの例外も考えられるため、情報を渡す方法としてrequestScopeに依存するべきではありません。データがリクエストから来ている場合は、それを仮パラメータとしてフローにマッピングしてください。

セキュリティ・ロール

タスク・フローが（呼び出されたタスク・フロー、ビュー・アクティビティ上で、または権限を評価するためのELを使用して）明示的な権限チェックを実行する場合は、いくつかの追加の考慮事項があります。タスク・フローがOracle ADFライブラリにパッケージ化される場合、現在、権限メタデータをコンポーネントの一部としてパッケージ化する方法はありません。このため、セキュリティを処理するために次の2つのアプローチのうちのいずれかが必要になります。

- 要件を文書化し、権限が含まれたポリシー・ファイルの断片をjarの一部としてパッケージ化します。消費側の開発者に、それらの権限を各自のjazn-data.xmlファイルに手動でマージしてもらいます。
- フロー・レベルでの権限を定義するために使用されるフローのパラメータを定義します。たとえば、その決定を行うELを使用して消費側の開発者によって移入される"isAdministrator"というタスク・フロー・パラメータを定義することが考えられます。次に、そのフロー内で、このローカライズされた権限プロキシを使用できます。

ファイル・アップロードの特殊なケース

af:inputFileコンポーネントを使用する断片ベースのバウンド・タスク・フローの場合、そのフローのコンシューマは、囲んでいるページのaf:formコンポーネントのusesUploadプロパティが"true"に設定されていることを保証する必要があります。

これは、もちろん、タスク・フローのコンシューマのために文書化する必要のある外部の依存関係です。

構成パラメータと接続

ほとんどの場合は、必要なすべての情報を仮パラメータとしてタスク・フローに渡してください。ただし、タスク・フローがweb.xmlや、weblogic-application.xmlなどのその他の構成ファイル内の接続、キュー名、コンテキスト・パラメータなどの、他の外部リソースに依存している可能性があることに注意してください。このような依存関係をすべて文書化する必要があります。

タスク・フローの操作にとって重要な構成パラメータの場合、パラメータまたはリソースが正しく設定されていないときは実行時例外を報告する必要があります。

リソース

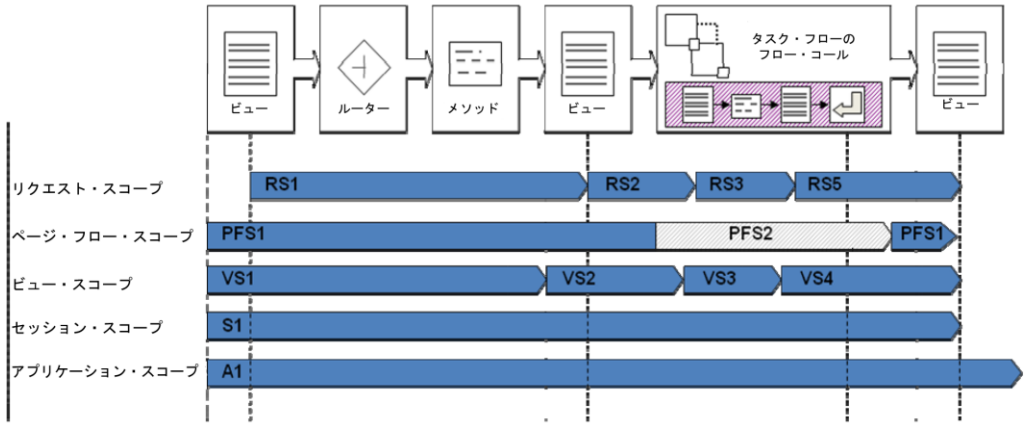
一般に、文字列や画像などのリソースはすべてタスク・フローのjar内にパッケージ化して、この点で自己完結型にする必要があります。ただし、複製したくない共通のエラー・メッセージやアイコンなどの共有リソースに、ある程度依存する可能性が必ず存在します。このような共有リソースは、親に組み込むのではなく、すべてのコンシューマから独立してパッケージ化する必要があります。それにより、親と子の両方のタスク・フローがこの共有リソース・ライブラリへの依存関係を継承および文書化できるようになります。

結論

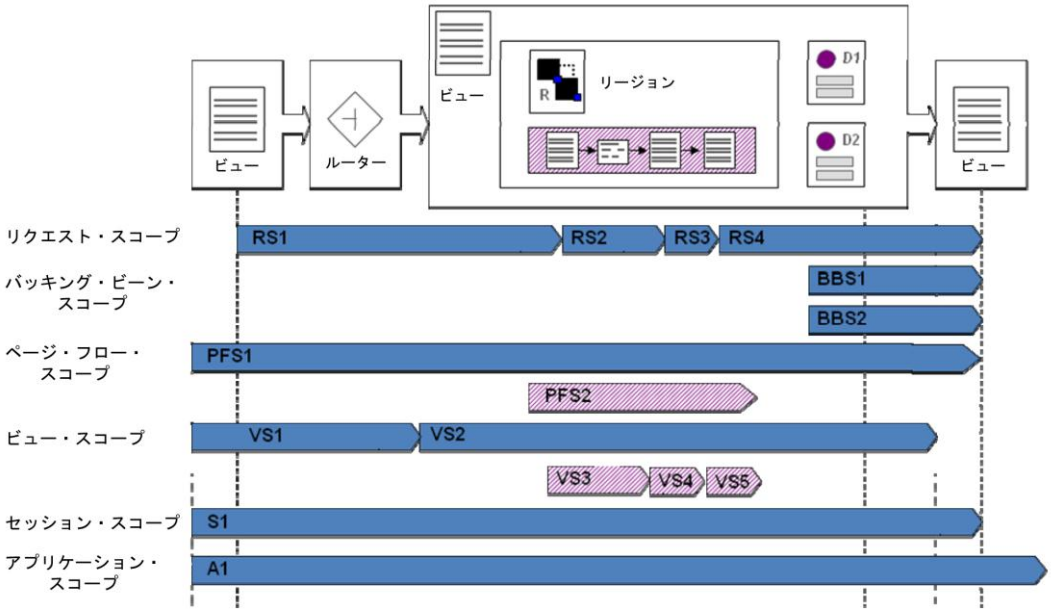
本書では、Oracle ADFコントローラのタスク・フローのための重要な実践および設計上の考慮事項のいくつかについて説明しました。共有で使用することを目的にタスク・フローを作成する場合は、ここで詳細に説明したさまざまなインタフェース・ポイントに細心の注意を払い、実行時の不正使用や意外性のリスクが最小限に抑えられるようにフローを慎重に強化および文書化する必要があります。

付録A - Oracle ADFコントローラとJSFメモリ・スコープ

完全なページ・フローのコンテキストにおけるスコープ



断片ベースのフローのコンテキストにおけるスコープ



タスク・フロー設計の基礎
2010年6月
著者：Duncan Mills
共著者：Frank Nimphius

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