Web アプリケーション フレームワーク チュートリアル

Sun Java™ Studio Enterprise 7 2004Q4

Sun Microsystems, Inc.
www.sun.com

Part No. 819-1290-10
2004 年 12 月, Revision A

原典: *Web Application Framework Tutorial*
Part No: 819-0727-10
Revision A

Please
Recycle

目次

はじめに

このチュートリアルでは、Web アプリケーションフレームワークのツールを使って Web アプリケーションを構築する際の仕組みとテクニックを紹介します。

Java 2 Platform, Enterprise Edition (J2EE™ プラットフォーム) を使った Web アプリケーションの構築については少なくとも多少の知識があるが、Web アプリケーションフレームワークを使って Web アプリケーションを構築したことがない開発者を読者対象にしています。

お読みになる前に

このチュートリアルは、Sun Java Studio Enterprise 7 開発環境 (以降、IDE と呼ぶ) で Web アプリケーションフレームワークツールを使用して作業を行う開発者の技術の習得に役立ちます。

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されている第三者の 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も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

次の章では、エンタープライズ Web アプリケーション開発用の Web アプリケーションフレームワークおよびツールセット (IDE) の主要機能の概要を説明しています。

- 第 1 章「チュートリアルを始める前に」

次の章では、Web アプリケーションフレームワークのツールを使って J2EE Web アプリケーションを構築する仕組みの概要を説明しています。

- 第 2 章「はじめに」

次の章では、以降のすべての章に必要なアプリケーションの基礎部分を作成し、これに Web アプリケーションフレームワークページの 1 ページ目を追加します。

- 第 4 章「1.1: アプリケーションインフラストラクチャ」
- 第 5 章「1.2: ログインページの作成」
- 第 6 章「1.3: ログインページのテスト実行」

次の章では、SQL ベースのモデルと、モデルのデータを表示するためのページを追加することによって既存のアプリケーションを拡張します。2 つのアプリケーションページをつなげて、データを調整、表示します。

- 第 7 章「2.1: SQL データベースアクセスのためのアプリケーションの準備」
- 第 8 章「2.2: CustomerModel の作成」
- 第 9 章「2.3: 顧客ページの作成」
- 第 10 章「2.4: 顧客ページのテスト実行」
- 第 11 章「2.5: 顧客ページへのログインページのリンク」
- 第 12 章「2.6: アプリケーションの実行」

次の章では、同じアプリケーション内の多数のボタンや HREF で再利用できるコマンドコンポーネントを作成します。これは、その親コンテナのビュークラス内でボタンまたは HREF の要求処理イベントに要求処理コードを実装するもう 1 つのテクニックです。

- 第 13 章「3.1: コマンドコンポーネントの作成」
- 第 14 章「3.2: 顧客ページへのログアウトリンクの追加」
- 第 15 章「3.3: ログイン/ログアウトコマンドコンポーネントのテスト実行」

次の章では、Web サービスベースのモデルと、モデルのデータを表示するためのページを追加することによって既存のアプリケーションを拡張します。Web サービス用のモデルを構築するには、Google 開発者の SDK に対する登録を行って、SDK をダウンロードする必要があります。

- 第 16 章「4.1: Web サービスモデルの作成準備」
- 第 17 章「4.2: Google 検索ページの作成」
- 第 18 章「4.3: Google 検索ページのテスト実行」
- 第 19 章「4.4: Google 検索ページへの結果リストの追加」
- 第 20 章「4.5: Google 検索ページのテスト実行」

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

チュートリアルを始める前に

J2EE Web アプリケーションフレームワークおよび Sun Java Studio Enterprise 7 ツールセット (IDE)、Web アプリケーションフレームワークへようこそ。これらはエンタープライズ Web アプリケーションの開発を行うツールです。

この章は、Web アプリケーションフレームワークの主な機能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の主な機能

Web アプリケーションフレームワークの主な機能は以下のとおりです。

- 複雑な設定が不要なターンキーの J2EE™ アプリケーション開発
- ハイパフォーマンスの実績ある J2EE フレームワーク実行時環境
- 完全なコンポーネントベースの開発
- 以下のグラフィカルアプリケーションビルダーツールセット
  - 論理アプリケーションツリーのエクスプローラビュー
  - アプリケーションコンポーネントと JSP 間での変更内容の自動同期
  - 高度なウィザード
- Web サービスモデルをサポート

Web アプリケーションフレームワークは、以下のような組織で使用されています。

- 中規模～非常に大規模なエンタープライズ Web アプリケーションを実行する大企業
- 金融、製造、政府、教育、医療、および通信業

Web アプリケーションフレームワークは、以下の点で有用なツールです。

- 経験の浅い Java™/J2EE 開発者を導き、手助けします。
  - 使いやすいグラフィカルな開発ツールで、すぐに習得できます。
  - 詳しい知識を持たない開発者でも複雑な J2EE API を利用できます。
  - 経験の浅い開発者でも、ハイパフォーマンスなエンタープライズアプリケーションを構築していく中で J2EE を習得できます。
- 経験豊かな Java/J2EE 開発者や設計者の作業を補完します。
  - 上級開発者は退屈な低レベルの J2EE 開発作業を避けて、その生産性を高めることができます。
  - アプリケーションアーキテクチャを拡張するための明確な開始点を設計者に提供します。
- J2EE API の世界への一歩を簡単に踏み出せるようにすることで、Web アプリケーションの開発とスキルやコンポーネントの再利用を促進します。

チュートリアルでは、Web アプリケーションフレームワークの機能を使って以下の作業を行う方法を示します。

- Web アプリケーションフレームワークアプリケーションの作成
- ページ (ViewBean と TiledView) および関連する JSP の作成
- モデル (JDBC™ SQL および WebService ベースのモデル) の作成と使用
- ページのリンク

第2章

はじめに

この章では、Web アプリケーションフレームワークツールを使って J2EE™ Web アプリケーションを構築する仕組みについて概説します。

この章は、以下の節で構成されています。

- ツールの紹介
- 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
- Web アプリケーションフレームワークチュートリアルについて

ツールの紹介

本書では、Web アプリケーションフレームワークツールを使って Web アプリケーションを構築する際の仕組みとテクニックを開発者に紹介していきます。

少なくとも既存の J2EE Web テクノロジ (サーブレットと JSP™ ページ) を使って Web アプリケーションを構築した経験は多少あるものの、Web アプリケーションフレームワークを使って Web アプリケーションを構築するのは初めて、という開発者を対象としています。

本書は、読者に Java の専門知識があること、および特定サーブレットコンテナの開発ならびに配備作業や使用されている開発ツールの経験があることを前提として記述されています。

Web アプリケーションフレームワークは最先端の設計パターンと一群のインタフェースから構成されていますが、本書の例では、Web アプリケーションフレームワーク実装の既存ベースクラスを拡張して特定アプリケーションオブジェクトを手動で構築するという、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作成するための最も基本的な方法のみを示します。これは、Web アプリケーションフレームワークアプリケーションを作成するための 1 つの方法にすぎません。

本書では、以下の 2 つの理由から、あまり高度なテクニックを示していません。1 番目の理由は、基本レベルから開始すると、初めてフレームワークを使用する開発者に Web アプリケーションフレームワークの仕組みを最も端的に伝えることができるからです。Web アプリケーションフレームワークを最大限に活用するには、フレームワークがアプリケーションとどう対話するかを正確に把握することがたいへん重要です。

2 番目の理由は、Web アプリケーションフレームワークアプリケーションを構築するためのさまざまな方法を完全に理解するには、基本的なテクニックを使ってアプリケーションを構築できることが必要不可欠な条件だからです。Web アプリケーションフレームワークを拡張して追加能力を付加する機能は、本書で示されているテクニックに基づいています。これらの基本的な例を理解すれば、こうした機能が Web アプリケーションフレームワークコアを拡張して補完する仕組みをさらに深く理解できます。そして、ときにはこれらを使用する代わりに、独自の Web アプリケーションフレームワーク拡張機能を自由に構築できるようになります (または、必要に応じてより基本的な方法を採用することもできます)。

したがって、このチュートリアルの最終的な目標は、フレームワーク上に構築されるアプリケーションと Web アプリケーションフレームワークとの対話を開発者が理解し、Web アプリケーションフレームワーク自体に習熟できるように、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構築するための最も基本的な方法を開発者に紹介することにあります。

Web アプリケーションフレームワークアプリケーションの作成

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、まずアプリケーション構造のレイアウトを設計してから、この構造に Web アプリケーションフレームワークオブジェクトを徐々に追加していくことで作成します。これはすべてゼロから手動で行うこともできますが、開発者の Web アプリケーションフレームワークアプリケーションの作成作業を手助けする Sun Java Studio Enterprise 7 用の Web アプリケーションフレームワークツールモジュールが作成されているため、この作業は簡素化されています。こうしたツールの助けにより、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、ウィザードを使って Web アプリケーションフレームワークコンポーネントを生成し、アプリケーション向けにこれらをカスタマイズする、というシンプルなプロセスで作成できます。

簡単な Web アプリケーションフレームワークアプリケーションの作成方法を示す前に、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構造の基礎について説明します。

J2EE/Web アプリケーションフレームワーク用語

このチュートリアルでは、アプリケーション、モジュール、コンポーネントといった用語が使用されています。こうした用語は、より一般的な Web アーキテクチャや開発の説明でも使用されているため、混乱を招く可能性があります。

このチュートリアルで使用されている最も重要な用語を、以下の表に示します。

用語	説明
*J2EE コンポーネント	J2EE アプリケーションコンポーネントと呼ばれることもあります。EJB、サーブレット、および Java Server Page™ (JSP™) といった J2EE サーバー上で配備、管理、実行される具体的なソフトウェアコンポーネントを示します。HTML やアプレットなども J2EE コンポーネントですが、Web アプリケーションフレームワーク Web アプリケーションには関係ありません。
*J2EE モジュール	J2EE アプリケーションを構成する基本単位を示します。J2EE モジュールは、1 つ以上の J2EE コンポーネントと 1 つのコンポーネントレベルの配備記述子から構成されています。J2EE モジュールは、スタンドアロンユニットとして配備することも、J2EE アプリケーション配備記述子と組み合わせて J2EE アプリケーションとして配備することもできます。サーブレットと JSP コンポーネント (またはそのいずれか) は、J2EE モジュールとしてパッケージ化されて、WAR ファイルとして配備されます。EJB コンポーネントは、J2EE モジュールとしてパッケージ化されて、JAR ファイルとして配備されます。任意の数の WAR ファイルと JAR ファイルを組み合わせて J2EE アプリケーションを形成し、EAR ファイルとして配備できます。WAR ファイル (J2EE Web アプリケーションとも呼ばれる J2EE モジュール) は、J2EE サーバー上に単独で配備できます。
*J2EE Web アプリケーション	J2EE サーブレットコンテナ (Web アプリケーションコンテナ) に配備することが可能な J2EE コンポーネントを含むスタンドアロンの J2EE モジュール。アプリケーションまたは J2EE アプリケーションという用語は、これが使用されている前後関係によっては、J2EE Web アプリケーションを意味することがあります。サーブレットや JSP から構成される J2EE モジュールを管理できるという点で、Sun Java System Application Server Standard Edition 7 2004Q2 や Apache Tomcat などは J2EE Web アプリケーションをサポートする製品ですが、これらの製品が EJB J2EE モジュールを含む可能性がある完全な J2EE アプリケーションを管理することはできません。

用語	説明
*J2EE アプリケーション	1 つ以上の J2EE モジュールと 1 つの J2EE アプリケーション配備記述子から構成され、Java アーカイブ (JAR) ファイル形式を使って .ear (enterprise archive: エンタープライズアーカイブ) ファイル名拡張子を持つファイルにパッケージ化されています。
Web アプリケーション フレームワークモジュール	Web アプリケーションフレームワークアプリケーション内のコンテンツとコンポーネントの論理パーティションと物理パーティションの両方を示します (J2EE モジュールと混同しないでください)。
Web アプリケーション フレームワークアプリケーション	非公式には、Web アプリケーションフレームワークアプリケーションは、Web アプリケーションフレームワークを使って作成された J2EE Web アプリケーションを示します。これは、最低 1 つの J2EE モジュール (Web アプリケーション) から構成されますが、他の標準 J2EE コンポーネントまたはモジュールを含む場合もあります。最小の 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、1 つの WAR ファイルから構成される J2EE Web アプリケーションです。公式には、Web アプリケーションフレームワークアプリケーションは、すべて同じサーブレットコンテキストで動作する関連する Web アプリケーションフレームワークモジュールのまとまりです。この意味では、Web アプリケーションフレームワークアプリケーションとは、Web アプリケーションフレームワークのこの論理的な抽象概念のみを意味します。

* この用語の詳しい説明は、『Java 2 Platform Enterprise Edition Specification v1.2 (J2EE)』の「J2EE8.1」の節を参照してください。

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編成

Web アプリケーションフレームワークは、形式的なアプリケーションおよびモジュールエンティティを提供します。Web アプリケーションフレームワークアプリケーションは、1 つ以上のサブパッケージ (Web アプリケーションフレームワークモジュール) を含む基本 Java パッケージです。アプリケーションが 1 つのモジュールだけから構成されることは完全に許容され、小規模アプリケーションにはよくあることです。各モジュールは、その親アプリケーションレベルのコンポーネントから動作を継承します。また、他のモジュールとは別個に、この動作をカスタマイズすることもできます。

J2EE Web アプリケーションコンテナでは、Web アプリケーションフレームワークアプリケーションは 1 対 1 でサーブレットコンテキストに対応するため、サーブレットコンテキストに対してコンテナが強制する制約を受けます。

アプリケーションの開発を始める前に、まずその編成を決定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アプリケーションにグループ化するモジュールを決定します。

  アプリケーションをいくつものモジュールに過度に分類するのは避けてください。なぜなら、Web アプリケーションフレームワークがこの機能を提供するためです。多くの場合、1 つのモジュールで十分です。

- アプリケーションパッケージ名を決定します。

  アプリケーションパッケージ名は複雑なものでもかまいませんが、たいていは、組織のパッケージ戦略を反映したものにします。それぞれのモジュールは、このアプリケーションパッケージの下のパッケージになります。

- 配備時の公開 Web アプリケーション名を指定します。

  Apache Tomcat では、`/webapps` ディレクトリ直下のディレクトリがこの名前になります。Sun Java System Application Server では、`$instance_dir/applications/j2ee-modules` ディレクトリ直下のディレクトリがこの名前になります。配備されるアプリケーション名は、WAR ファイルの名前と同じになります。

たとえば、Web アプリケーションフレームワークアプリケーションが 2 つのアプリケーションフレームワークモジュール (「module1」と「module2」) から構成されていて、このアプリケーションに「myapp」という名前を付けたと仮定します。その場合、完全なアプリケーションパッケージ名は `com.mycompany.myapp` になります。

- アプリケーションパッケージ: `com.mycompany.myapp`
- 「module1」パッケージ: `com.mycompany.myapp.module1`
- 「module2」パッケージ: `com.mycompany.myapp.module2`

一般に、アプリケーションパッケージ名には、そのモジュール名とは別の名前を付ける必要があります。

たとえば、当初、思い付きでアプリケーションおよびその主要モジュールの両方に「foo」という名前を付けたとします。こうした命名は、アプリケーションやアプリケーション開発ツールを理解しようとするときに容易に混乱の原因になる危険性があります。この場合は、アプリケーションパッケージに「fooapp」といった名前を付けたり、主要モジュールに「main」あるいは「module1」などの名前を付けてください。これによってアプリケーション構造がはるかに分かりやすくなります。特に、将来アプリケーション構造に追加を行う場合に役立ちます。

Web アプリケーションフレームワーク チュートリアルについて

Web アプリケーションフレームワークとそのツールの使い方を学習するために、これから簡単なアプリケーションを開発していきます。このアプリケーションは、ログインページと顧客アカウントページという 2 つのページから構成されており、以下を行います。

- ユーザーが送付したフィールド値を取り出す。
- ユーザーに状態メッセージを返す。
- QueryModel を使って顧客情報を取り出す。
- QueryModel を使って顧客情報を更新する。
- ユーザー入力を QueryModel SQL WHERE 条件と調整する。
- あるページから別のページに移動する。
- WebServiceModel を使って、Google インターネット検索を実行する。
- WebServiceModel の複数検索結果を表示する。

このチュートリアルはいくつもの節に分かれ、アプリケーション開発に必要な作業を課します。各節では広範な内容を説明します。各節を終えると、実行可能なアプリケーションが完成するようになっています。

1 つの章で説明されている作業はどちらかというとそれぞれ独立した作業で、詳細におよぶ作業もいくつか含まれています。

第3章

チュートリアルの各節 (参照用)

この章では、この『Web アプリケーションフレームワーク チュートリアル』に含まれる節について概説します。

次の各節について、各節で行う作業を参照のために示します。

- 1.1～1.3
- 2.1～2.6
- 3.1～3.3
- 4.1～4.5

1.1～1.3

1.1 ～ 1.3 では、後続のすべての章で必要になるアプリケーションインフラストラクチャを作成して、最初の Web アプリケーションフレームワークページを追加します。

- 1.1

  作業 1: 新規 Sun Java Studio Web アプリケーション

- 1.2

  作業 2: ログインページの作成

- 1.3

  作業 3: ログインページのテスト実行

2.1～2.6

2.1～2.6 では、SQL ベースのモデルと、そのモデルのデータを表示するページを追加することで、既存アプリケーションを拡張します。それから、この 2 つのアプリケーションページに同期されたデータが表示されるように、2 つのページをリンクさせます。

- 2.1

  作業 1: SQL データベースへのアクセス

- 2.2

  作業 2: CustomerModel の作成

- 2.3

  作業 3: 顧客ページの作成

- 2.4

  作業 4: 顧客ページのテスト実行

- 2.5

  作業 5: ログインページの顧客ページへのリンク

- 2.6

  作業 6: アプリケーションの実行

3.1～3.3

3.1～3.3 では、同じアプリケーション内の多数のボタンや HREF で再利用できるコマンドコンポーネントを作成します。これは、その親コンテナのビュークラス内でボタンまたは HREF の要求処理イベントに要求処理コードを実装するもう 1 つのテクニックです。

- 3.1

  作業 1: コマンドコンポーネントの作成

- 3.2

  作業 2: 顧客ページへの HREF の追加

- 3.3

  作業 3: ログイン/ログアウトコマンドのテスト実行

4.1～4.5

4.1～4.5 では、Web サービスベースのモデルと、そのモデルのデータを表示するページを追加することで、既存アプリケーションを拡張します。Web サービスのモデルを構築するには、Google の開発者用 SDK に登録して、これをダウンロードする必要があります。

- 4.1

  作業 1: Web サービスのユーザー登録とダウンロード

- 4.2

  作業 2: Google 検索ページの作成

- 4.3

  作業 3: Google 検索ページのテスト実行

- 4.4

  作業 4: TiledView ページレットの作成

- 4.5

  作業 5: 結果を使った Google 検索ページのテスト実行

第4章

1.1: アプリケーションインフラストラクチャ

この章では、以降のすべての作業で必要とされる Sun Java Studio Web アプリケーションフレームワーク (Web アプリケーションフレームワーク、App フレームワーク、SJSAF、および JATO とも呼ばれます) アプリケーションインフラストラクチャの作成方法を説明します。

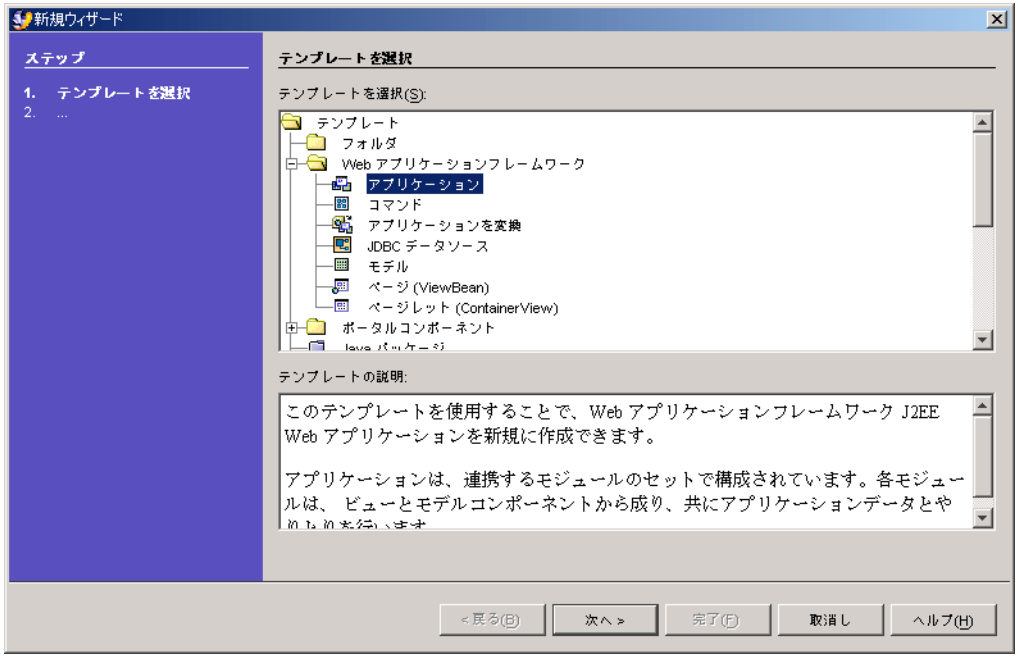
作業 1: 新規 Sun Java Studio Web アプリケーション

ページを開発する前に、Web アプリケーションフレームワークアプリケーションインフラストラクチャ (WAR ディレクトリ構造とサポートするファイル) を作成する必要があります。これは、それぞれの Web アプリケーションフレームワークアプリケーションで 1 度行う必要があります。

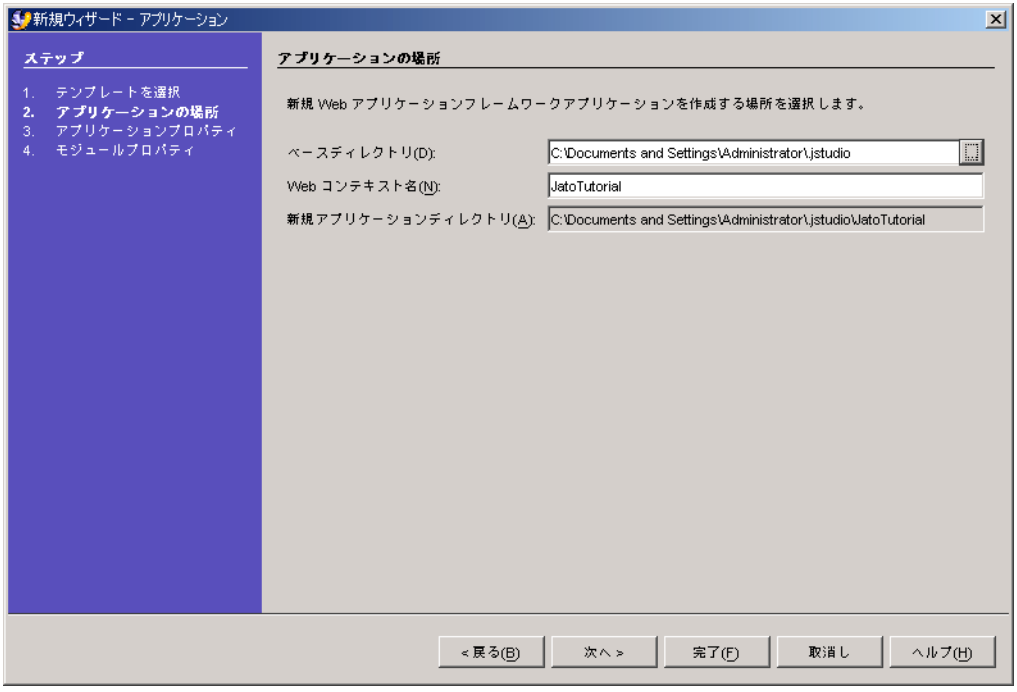
アプリケーションウィザードの作成

アプリケーションを作成する前に、アプリケーションの配置場所を決定する必要があります。ターゲットの実行環境に配備しなくてもアプリケーションをテストできるように、通常はサーブレットコンテナの `webapps` ディレクトリで直接アプリケーションを開発します。すでに Sun Java Studio (Studio) は使用しているため、任意の場所にアプリケーションを配置して、組み込み型の Sun Java System Application Server モジュールを使って同じ場所でテストできます。

1. Sun Java Studio Enterprise 7 メニューオプションの「ファイル」>「新規 Web アプリケーションフレームワークアプリケーション」を選択します。
   「テンプレートを選択」パネルが表示されます。



2. Web アプリケーションフレームワークフォルダを展開します。
3. アプリケーションを選択します。
4. 「次へ」をクリックします。
   「アプリケーションの場所」パネルが表示されます。



デフォルトのベースディレクトリは Sun Java Studio `user-dir` になりますが、これはこの例で示されているものとは異なる場合もあります。任意の既存ディレクトリを Web アプリケーションフレームワークアプリケーション用のベースディレクトリとして選択できます。

注 – 多くの開発者は、アプリケーションを配備するサーブレットコンテナの `webapps` ディレクトリを使用します。Sun Java Studio を使って Web アプリケーションフレームワークアプリケーションを実行する方法を、後でこのチュートリアルで示します。したがって、Web アプリケーションは、任意の場所に配置できます。

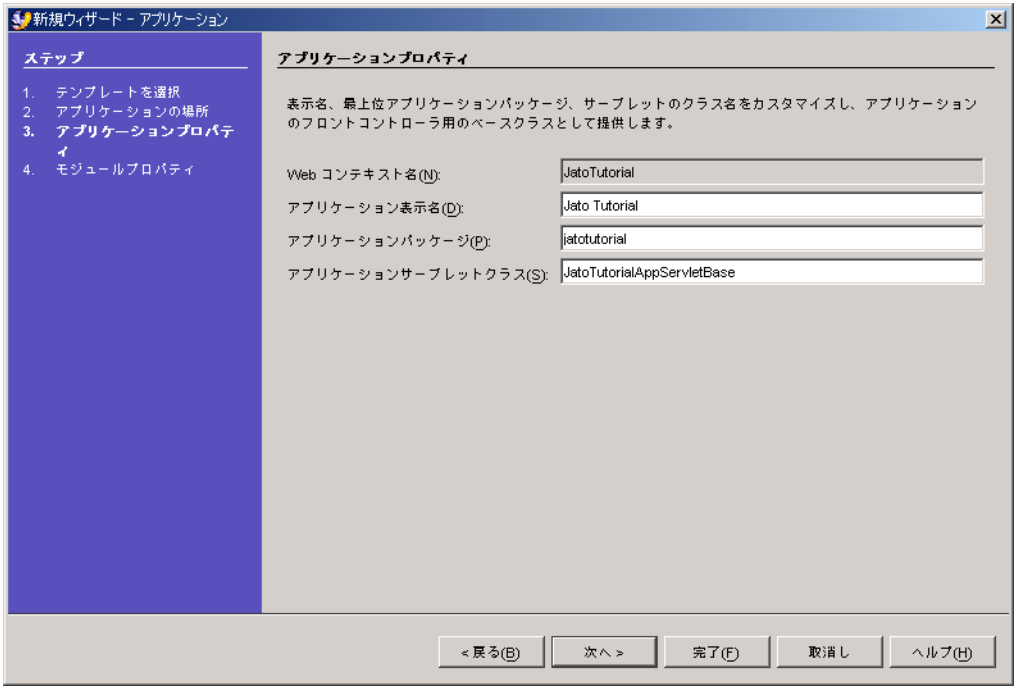
---

5. **「Web コンテキスト名」フィールドに `JatoTutorial` と入力します。**

   「ベースディレクトリ」フィールドと「コンテキスト名」フィールドに入力すると、「新規アプリケーションディレクトリ」フィールドの値が設定されます。

6. **「次へ」をクリックします。**

   「アプリケーションプロパティ」パネルが表示されます。

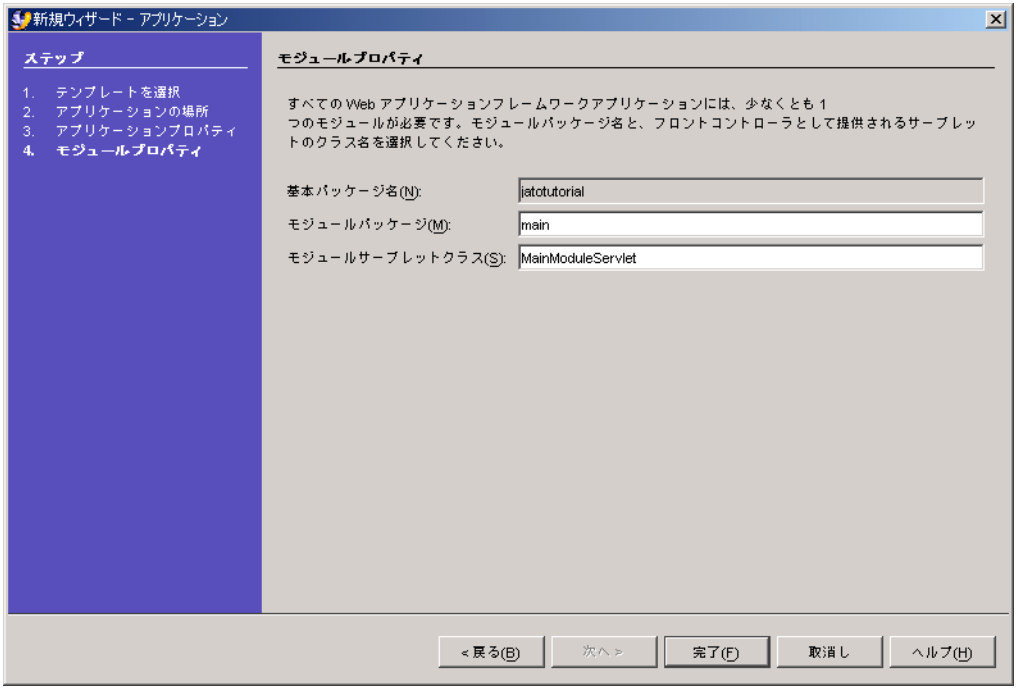


このパネルのフィールドは、前のパネルの「Web コンテキスト名」フィールドの値を使って設定されます。

このチュートリアルでは、デフォルト値を使用します。

7. **「次へ」をクリックします。**

「モジュールプロパティ」パネルが表示されます。



このチュートリアルでは、デフォルト値を使用します。

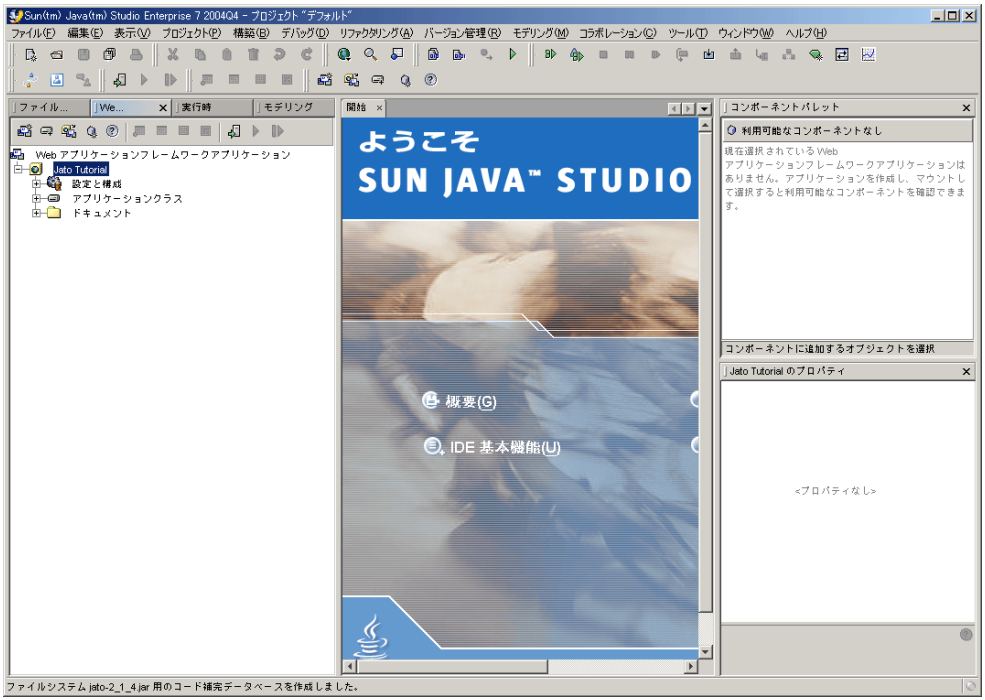
8. **「完了」をクリックします。**

9. **ポップアップダイアログを閉じます。**

   アプリケーションが作成されます。

注 – 処理時間は使用するマシンによって異なります。

新規アプリケーションが IDE のエクスプローラウィンドウの Web アプリケーションフレームワークアプリケーションツリーに「Web アプリケーションフレームワークアプリケーション」というラベルを付けて表示されます。



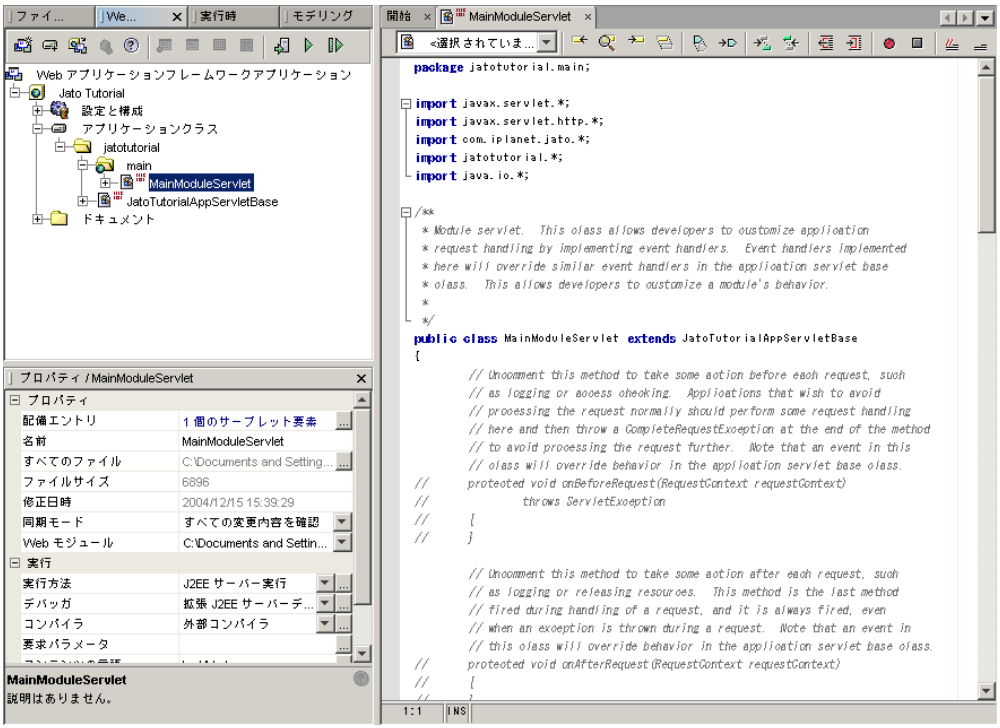
注 – 「了解」をクリックすると、上の画面が表示されます。特に IDE をインストールしたばかりの場合、この時点では「プロパティ」ウィンドウが表示されていないことがあります。ウィンドウの表示を切り替える場合は、「ウィンドウ」メニューから表示または非表示にするウィンドウを選択します。

---

10. **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エクスプローラタブの「アプリケーションクラス」ノードを展開して、アプリケーションレイアウトを表示し、作成された 2 つのサーブレットクラスのコードを確認します。**
    - JatoTutorialAppServletBase
    - MainModuleServlet

11. **「プロパティ」ウィンドウが、意図した場所に表示されないことがあります。新しくインストールした IDE 内でウィンドウを移動させるには、ウィンドウのタイトルバーを選択し (タイトルバーマウスの左ボタンを押したままにすると選択できます)、目的の位置までドラッグします。**

    ウィンドウをドラッグすると、赤い外枠が表示されて移動先の位置と表示区域が確認できます。ウィンドウは画面の四隅に合わせたり、他のウィンドウにタブ付きパネルとして合体させたりできます。次の図は、エクスプローラビューの左下に「プロパティ」ウィンドウを表示した例です。



アプリケーションサーブレット

アプリケーションサーブレットの JatoTutorialAppServletBase は、アプリケーションの全モジュールサーブレットのスーパークラスになる、という以外には、アプリケーションに対して特に何の意味も持ちません。

Web アプリケーションフレームワークモジュールサーブレットに装備のイベントを実装すると、セッションと要求のライフサイクルをカスタマイズして制御できます。

以下に例を示します。

- onNewSession
- onSessionTimeout
- onBeforeRequest
- onAfterRequest

一般に、同じアプリケーション内のあらゆるモジュールサーブレットは、これらの全イベントに同じ動作を要求します。このため、こうしたイベントの動作は、すべてのモジュールサーブレットが拡張できるクラスに実装することを推奨します。

ただし、技術的に言えば、アプリケーションサーブレットは必要ありません。モジュールサーブレットの階層は、それが Web アプリケーションフレームワークの com.iplanet.jato.ApplicationServletBase ファイルから派生している限り、カスタマイズできます。

このアプリケーションはモジュールを 1 つだけ持ち、定義では、モジュールサーブレットは 1 つになります。このため、アプリケーションサーブレットの役割は、マルチモジュールアプリケーションの場合ほど有用ではありません。

モジュールサーブレット

モジュールサーブレットの MainModuleServlet は、すべての要求に対して呼び出される実際のサーブレットです。アプリケーションに対するすべてのアクセスは、この「フロントコントローラ」サーブレットを通ってから、適切な「要求ハンドラ」クラス (このチュートリアルで後で実装) に制御が渡ります。

このクラスには、多くのコードは必要とされません。必要な要求処理コードは、すべて Web アプリケーションフレームワークの com.iplanet.jato.ApplicationServletBase ファイルに格納されています。上級開発者は、com.iplanet.jato.ApplicationServletBase クラスのソースコードを調べることで、要求が処理される仕組みを把握してください。

高度なヒント - モジュール

「main」モジュールフォルダを選択すると、そのプロパティが Studio エクスプローラウィンドウ下部のプロパティシートに反映されます。このとき「モジュール」プロパティが「True」になっていることに注意してください。これを「False」に変更すると、このモジュールは通常フォルダまたはパッケージとなり、web.xml ファイル (標準 Web アプリケーション構成ファイル) の MainModuleServlet 用のエントリが削除されます。

どの通常フォルダも、右クリックして「モジュールに変換」アクションを選択することで、Web アプリケーションフレームワークモジュールに変換できます。続いて、モジュールサーブレットにする Java サーブレットクラスをそのフォルダから選択するように指示されます。選択する代わりに、名前を入力して新規作成することもできます。

第5章

1.2: ログインページの作成

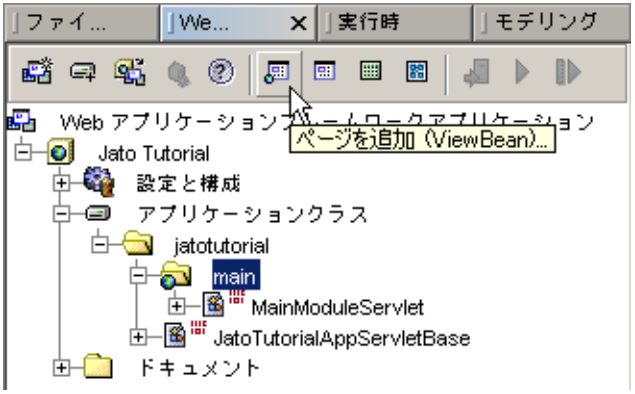
この章では、作成したアプリケーションインフラストラクチャに最初の Web アプリケーションフレームワークページを追加する方法を説明します。

作業 2: ログインページの作成

アプリケーションの最初のページの作成

ViewBean の追加

1. 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エクスプローラタブで「main」モジュールフォルダを選択します。



2. Web アプリケーションフレームワークツールバーの「ページを追加 (ViewBean)」ボタンをクリックします。

   または

   a. IDE のメニューオプションの「ファイル」>「新規」を選択します。

   b. 「Web アプリケーションフレームワーク」ノードを展開します。

   c. 「ページ (ViewBean)」を選択します。

   d. 「次へ」をクリックします。

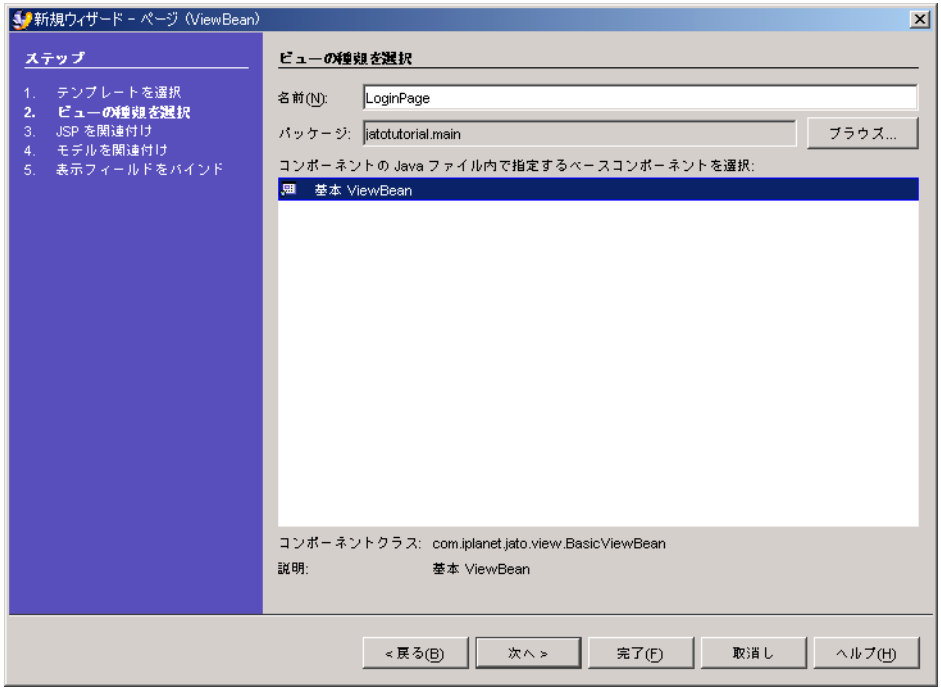
   または

   a. 「main」モジュールフォルダを右クリックします。

   b. 「追加」を選択します。

   c. 「ページ (ViewBean)」を選択します。

   「ビューの種類を選択」パネルが表示されます。

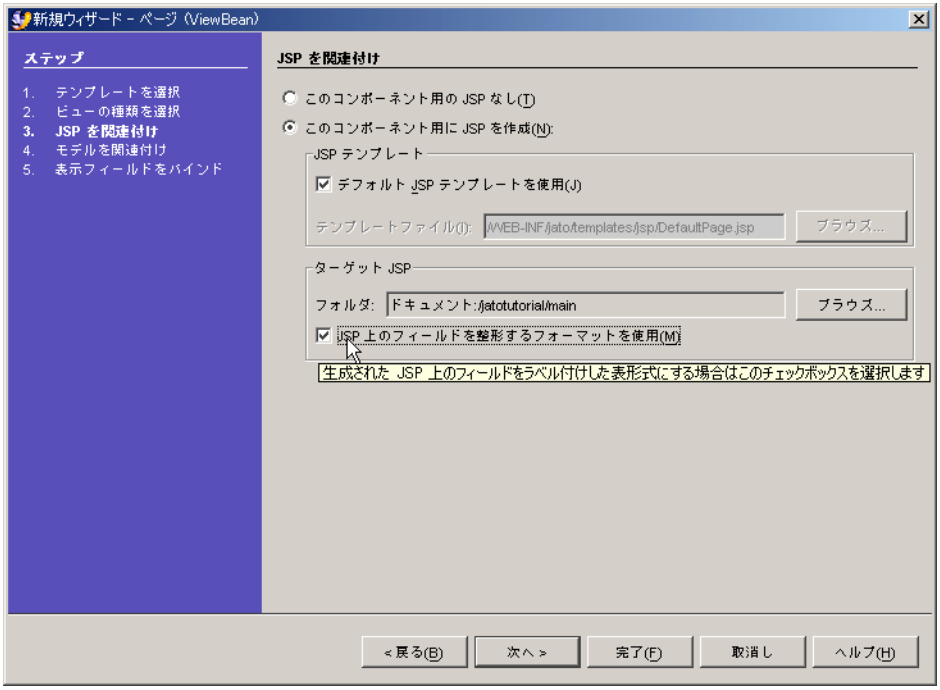


3. 「名前」フィールドに LoginPage と入力して、<デフォルト名> を置き換えます。

4. ベースコンポーネントリストから、「基本 ViewBean」を選択します。

5. 「次へ」をクリックします。

「JSP を関連付け」パネルが表示されます。

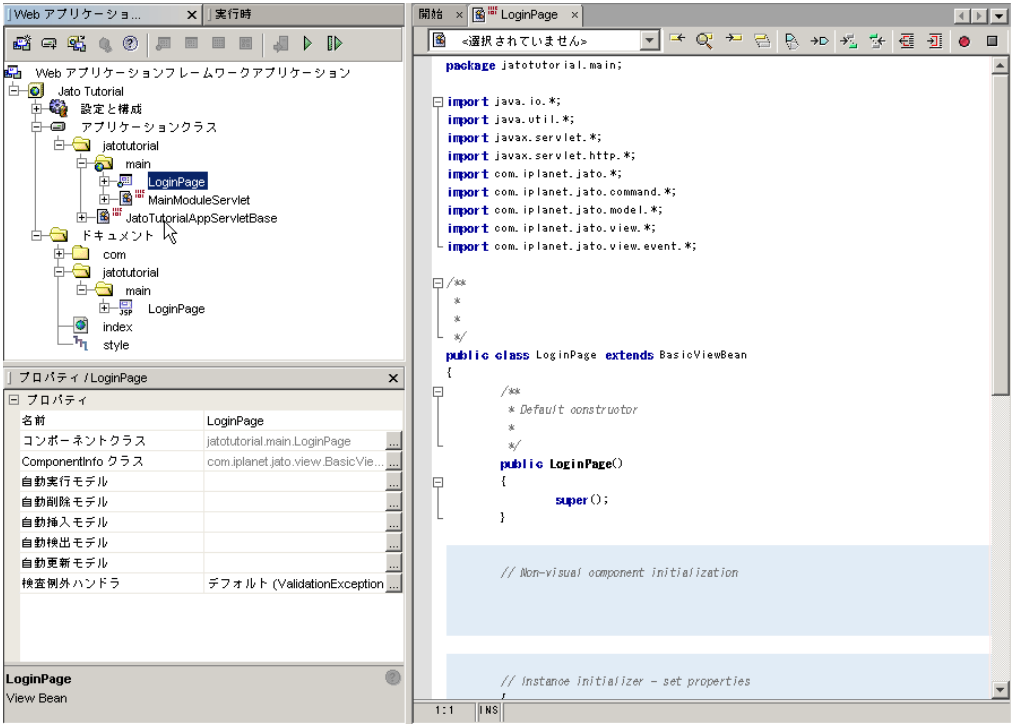


6. すべてのデフォルトを受け入れますが、オプションの「JSP 上のフィールドを整形するフォーマットを使用」チェックボックスを選択します。

7. 「完了」をクリックします。

ViewBean が作成されます。

注 – ページウィザードにはその他の手順もあります。ただし、これらの作業では、LoginPage では不要なモデルフィールドのバインドを行います。これら手順は、後の作業で使用します。



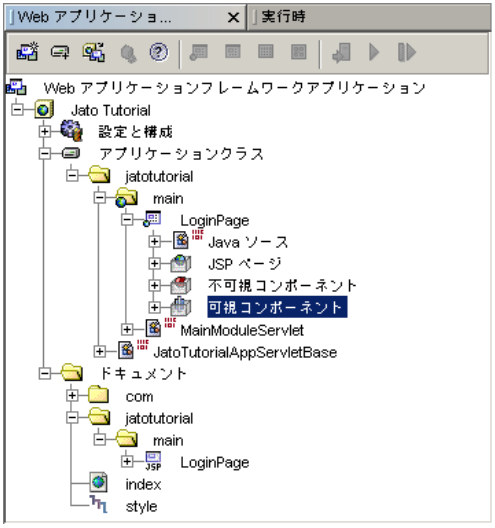
8. 「LoginPage」をダブルクリックします。

   生成されたソースコードが IDE のソースエディタに表示されます。

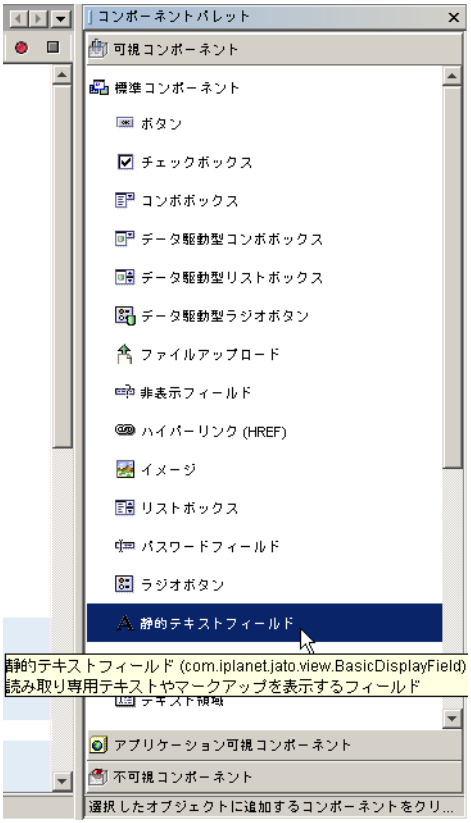
注 – LoginPage の作成時に JSP を作成することにしたため、ViewBean のパッケージ構造を反映するディレクトリ構造 (/jatotutorial/main) の「ドキュメント」フォルダに JSP が追加されています。便宜上、LoginPage を使用する JSP へのリンクは、「LoginPage」ノードの下の JSP ページのノードに配置されています。

ログインページへの表示フィールドの追加

1. 「LoginPage」ノードを展開します。
2. 「LoginPage」ノードの下の「可視コンポーネント」ノードを選択します。

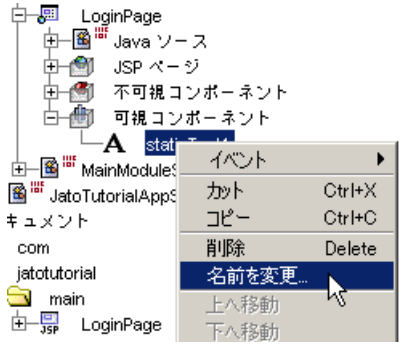


3. Web アプリケーションフレームワークコンポーネントパレットで、「静的テキストフィールド」オプションをクリックします。



静的テキストの可視コンポーネントが「可視コンポーネント」ノードに追加されます。

デフォルト名は *staticText1* です。



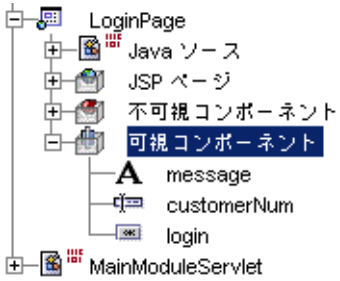
4. 「staticText1」フィールド名を右クリックします。

5. 「名前を変更」を選択します。
6. フィールド名を「message」に変更します。
7. さらに 2 つの表示フィールドを追加します。

   以下の表は、2 つの可視コンポーネントの種類とその名前、および「ボタン」の種類と初期値を示しています。

種類	名前	初期値
テキストフィールド	customerNum	
ボタン	login	オブジェクトの種類 : 文字例 オブジェクト値 : Login

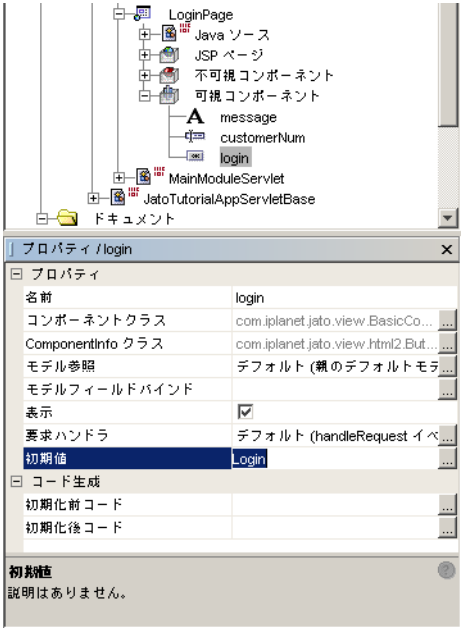
   3 つの表示フィールドが、「LoginPage」の「可視コンポーネント」ノードの下に表示されます。



   ページに表示フィールドを追加すると、このページを使用する JSP に、表示フィールド用の適切な JSP タグも追加されます。

8. 「login」ノードを選択して、ボタンの「初期値」プロパティを設定します。
9. 「初期値」プロパティ値の入力領域をクリックします。
10. 「Login」という文字列を入力します。

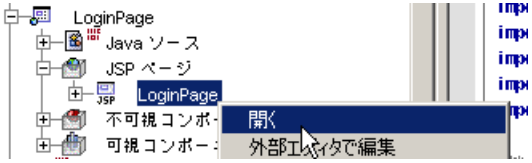
    ボタンの値は、ブラウザのボタンに表示される文字列になります。



11. LoginPage の JSP を開いて、3 つの表示フィールドのタグをチェックします。

    a. 「LoginPage」ノードの下の「JSP ページ」ノードを展開します。

    b. 「LoginPage」JSP をダブルクリックして、IDE のソースエディタでこれを開きます。



12. 必要に応じて JSP レイアウトを書式設定します。

注 – ページウィザードで JSP ページのコンテンツを整形するオプションを選択しているため、すでに一部の基本的な HTML 書式が設定されている可能性があります。正確には、自動的に提供される追加の書式は、親のビューコンポーネントに追加される子のビューコンポーネントの種類と条件によって異なります。すでに手動で単純な可視コンポーネントを加えたように、追加の書式が提供されないために若干の変更が必要になる場合があります。

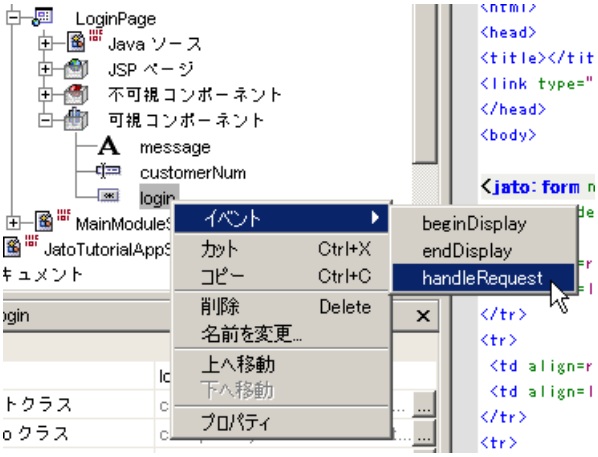
これは IDE のソースエディタで直接編集することも、任意の WYSIWYG HTML エディタを使って編集することもできます。

以下に、JSP に最小限の変更を加えた例を示します (関連するコードのみを示します)。以下の一部の HTML ソースコードは、強調するために太字で示されています。

```
<jato:form name="LoginPage" method="post">
<table border=0 cellspacing=2 cellpadding=2 width="100%">
<tr>
  <td align=right valign=middle width="20%"></td>
  <td align=left valign=middle><jato:text name="message"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Customer Num:</b></td>
  <td align=left valign=middle><jato:textField name="customerNum"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"></td>
  <td align=left valign=middle><jato:button name="login"/></td>
</tr>
</table>
</jato:form>
```

「Login」ボタンへのコードの追加

1. 「login」ボタンを右クリックします。
2. 「イベント」>「handleRequest」を選択します。



LoginPage.java ファイルが開き、handleLoginRequest メソッドスタブが挿入されます。

3. ログインボタンで handleRequest コードを実行するようにします。

   次のデフォルトのコードを、この後に示すコード例に置き換えます。

```
getParentViewBean().forwardTo(getRequestContext());
```

```
public void handleLoginRequest(RequestInvocationEvent event) {
    // 顧客番号を取り出す
    String custNum = getDisplayFieldStringValue(CHILD_CUSTOMER_NUM);

    String theMessage = "";

    // 顧客番号を確認
    if (custNum.equals("1") || custNum.equals("777") ||
            custNum.equals("410")) {
        theMessage = "Congratulations, " + custNum +
                ", you are now logged in!";
    } else {
        theMessage = "Sorry, " + custNum +
                ", your customer number was incorrect!";
    }

    // 出力状態メッセージを設定
    getDisplayField(CHILD_MESSAGE).setValue(theMessage);

    // 現在のページを再表示
    forwardTo();
}
```

第6章

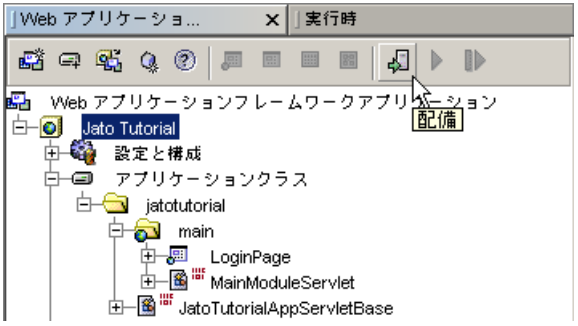
1.3: ログインページのテスト実行

この章では、Web アプリケーションフレームワークアプリケーションの実行方法を説明します。

作業 3: ログインページのテスト実行

Web アプリケーションのコンパイル

1. 「Jato Tutorial」ノードを選択します。



2. **エクスプローラウィンドウ上部の Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。**

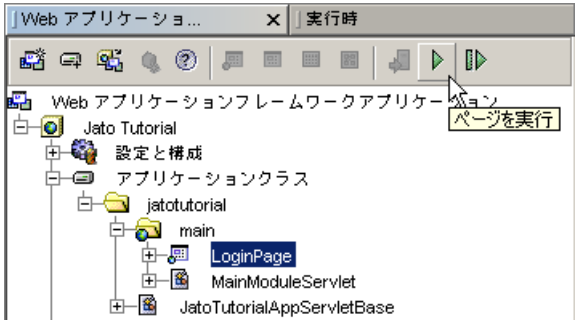
   この操作を行うだけで、Web アプリケーション全体 (コンパイルが必要なクラス) がコンパイルされて、Sun Java System Application Server に配備されます。

   チュートリアルのすべての指示に従えば、Web アプリケーションはエラーなしでコンパイルされて配備されます。エラーメッセージについては、IDE の「出力」ウィンドウを参照してください。

   Web アプリケーションのいずれかのリソースに変更を加え、これを Sun Java System Application Server (Application Server) で実行する場合は、この手順で配備することが必要です。

ログインページのテスト実行

1. 「LoginPage」を選択します。



2. **エクスプローラウィンドウ上部の Web アプリケーションフレームワークツールバーの「ページを実行」ボタンをクリックします。**

注 – 「ページを実行 (再配備)」ボタン (Web アプリケーションフレームワークツールバーの「実行」ボタンの右側) をクリックすると、Sun Java System Application Server による全リソース (JSP やクラスなど) の再読み込みを強制実行できます。一部のブラウザでは、アプリケーションのページを再実行する前に、ブラウザの全インスタンスを閉じる必要があります。

デフォルトのブラウザでアプリケーションが起動します。

ログインの成功のテスト

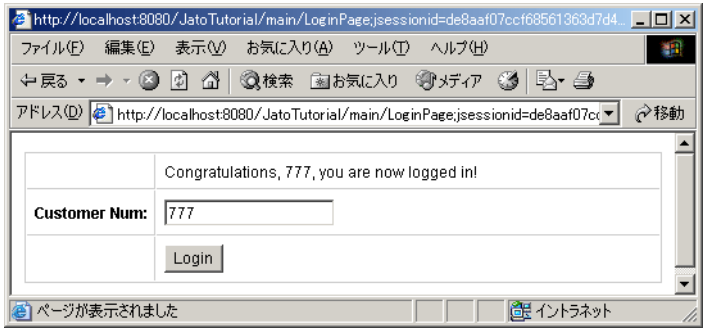
1. 有効なログイン名を入力します (たとえば、1、777、または 410 は、ハードコードされた有効な顧客番号です)。
2. 「Login」をクリックします。

注意 – Microsoft Internet Explorer の特定のバージョンを含む一部の Web ブラウザでは、テキストフィールドで Enter キーを押すと、自動的にフォームが送付されます。ただし、サーバーはこの送付アクションからどのボタンに対応すべきかが分かりません。<jato:form> タグは、デフォルト時に起動すべきボタンをサーバーに伝えるための属性「defaultCommandChild」を提供しています。

この機能の詳細は、タグライブラリのマニュアルを参照してください。

今の段階では、ボタンを直接クリックしてください。

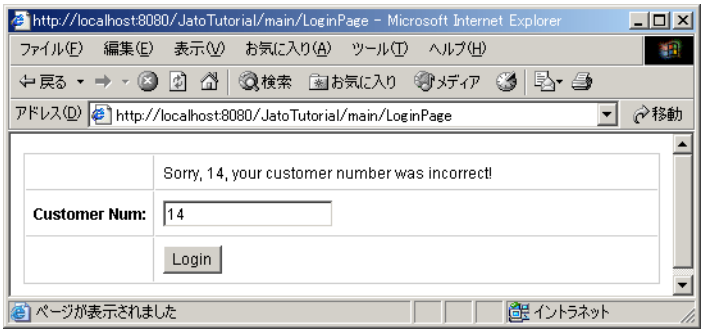
ログインページが再描画されて、成功を示すメッセージが表示されます。



ログインの失敗のテスト

1. 無効なログイン名を入力します (たとえば、`foo`、`8`、または `14` など、前述のハードコードされた有効な顧客番号以外のもの)。
2. 「Login」をクリックします。

   ログインページが再描画されて、失敗を示すメッセージが表示されます。



代替実行環境

1. IDE の外部でアプリケーションをテスト実行する場合は、アプリケーションをコンパイル、パッケージ化して WAR ファイルを作成し、その WAR ファイルを `webapps` ディレクトリに配置します (このディレクトリはコンテナにより異なりますが、多くの場合は `webapps` という名前になります)。

2. PointBase ドライバファイルをサーブレットコンテナのクラスパスに追加する必要があります。ドライバは、以下の IDE のインストールディレクトリに格納されています。

   Windows の場合: `<studio-install-dir>\AppServer7\pointbase\client_tools\lib\pbclient42RE.jar`

   Solaris/Linux の場合: `<studio-install-dir>/AppServer7/pointbase/client_tools/lib/pbclient42RE.jar`

   これを行うための最も簡単な方法は、アプリケーションの `WEB-INF/lib` ディレクトリにこのドライバをコピーすることです。

3. ブラウザを開き、サーブレットコンテナに適切な URL を使ってこれを実行します。

   この変形として可能なのは、URL 末尾にページ名 (LoginPage) を付加することです。

   Apache Tomcat または Caucho Resin サーブレットコンテナの場合
   `http://localhost:8080/JatoTutorial/main/LoginPage`

注 – この作業は、チュートリアルの途中で必要になったときに再度参照してください。

第7章

2.1: SQL データベースアクセスのためのアプリケーションの準備

この章では Web アプリケーションフレームワークアプリケーションを拡張して、SQL データベースにアクセスする準備を行う方法を説明します。

SQL ベースのモデルとそのモデルのデータを表示するページを追加することで、既存アプリケーションを拡張します。

2 つのアプリケーションページで同期されたデータが表示されるように、2 つのページをリンクさせます。

作業 1: SQL データベースへのアクセス

サンプルデータベースへの接続

このチュートリアルの以降の記述では、より広範な Web アプリケーションフレームワークの機能を使用するのに必要となる RDBMS データベースがあることを前提としています。

Web アプリケーションフレームワークアプリケーションから RDBMS にアクセスするための前提条件はありません。したがって、作成するアプリケーションが RDBMS にアクセスしないこともあります。ただし、その場合でも、他のエンタープライズシステムの準備、設定、および接続が必要なことがあります。

以降の手順は、IDE で事前構成済みの PointBase データベースを利用することを前提としています。このデータベースは、事前構成済みの JDBC 接続を使って IDE がサンプルデータベースに接続しようとする場合に自動的に起動します。また、自動的にインストールされた Sun Java System Application Server には、事前構成済みデータベース接続リソースが含まれています。新規アプリケーションそれぞれは、あらかじめ用意されている PointBase サンプルデータベースの接続リソースに一致したデータ

ソースで事前構成されます。この方法を使うことで、開発者は特に自分で構成作業を行わなくても Web アプリケーションフレームワークと PointBase サンプルデータベースを素早く利用することができます。

JDBC データソース

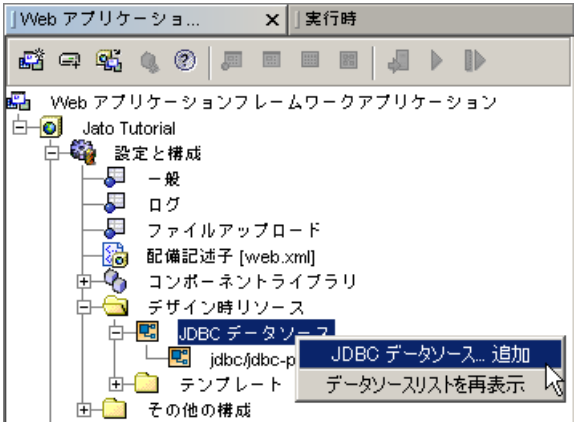
JDBC データソースは、Web アプリケーションフレームワークの「JDBC データソース」ウィザードを使って作成できます。

ただし、IDE とともに提供されている PointBase サンプルデータベースをポイントするものがデフォルトで作成されています。

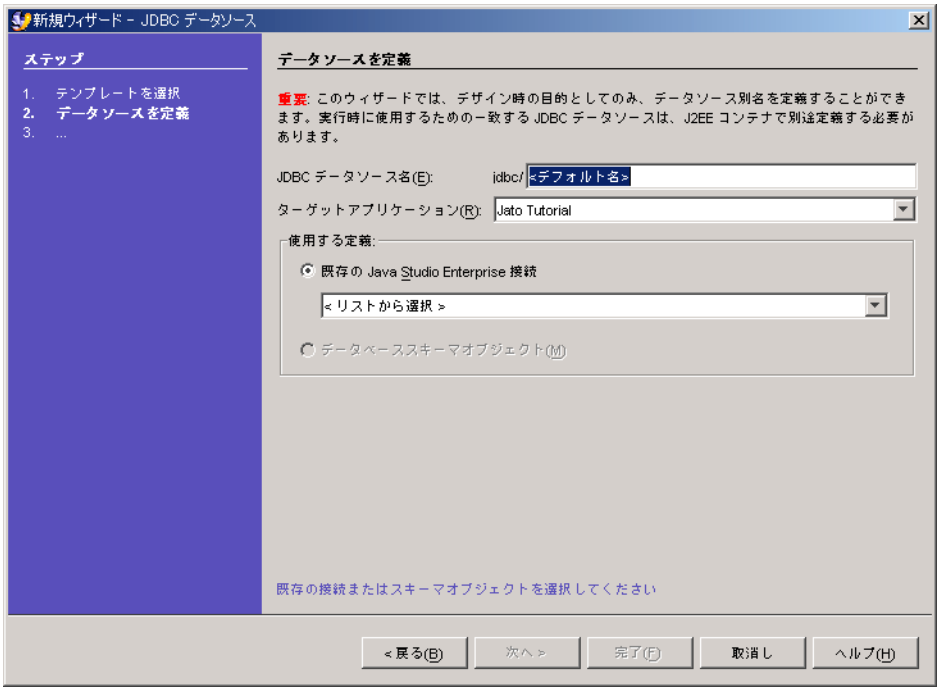
チュートリアルで使用されているデータベース以外の JDBC データソースを作成する必要がある場合は、以下の手順に従ってください。

(この必要がない場合は、この内容を習得するために以降を読むか、または次の「Tomcat (および他の非 JNDI コンテナ) SQL 接続の準備」の項に進んでください。)

1. 「Web アプリケーションフレームワーク Web アプリケーション」ノード (「Jato Tutorial」) の下の「設定と構成」フォルダを展開します。
2. 「デザイン時リソース」フォルダを展開します。
3. 「JDBC データソース」ノードを右クリックします。
4. 「JDBC データソース...追加」を選択します。



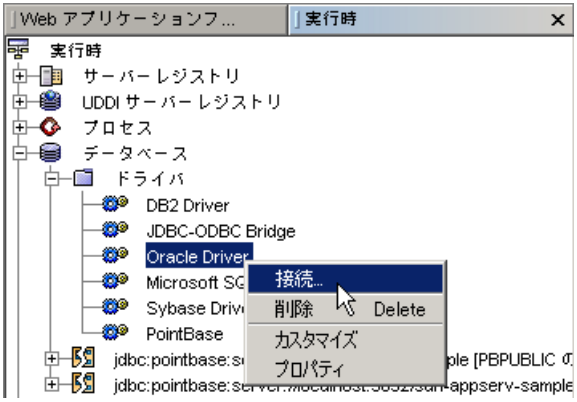
「JDBC データソースを定義」パネルが表示されます。



5. 「JDBC データソース名」テキストボックスに、使用するデータソース名を入力します。

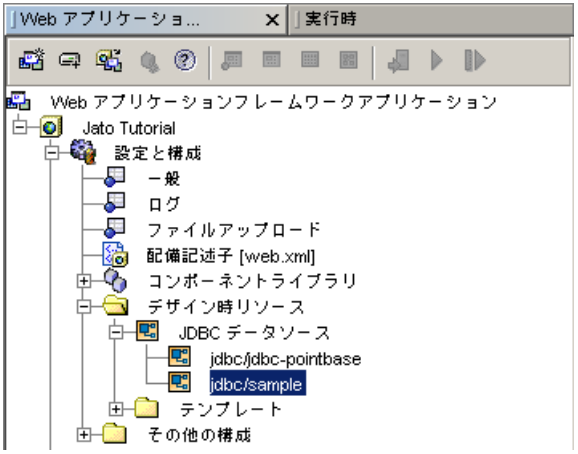
6. 「<リストから選択>」コンボボックスで、適切な JDBC 接続を選択します。

   必要な接続が存在しない場合は、これを作成する必要があります。これは、Web アプリケーションフレームワークツールの適用範囲外のツールによって実行します。「実行時」ウィンドウを選択し、「データベース」ノードを展開して「ドライバ」ノードを展開します。目的の「ドライバ」ノードをマウスの右ボタンでクリックして、「接続...」を選択します。接続を追加する前に、データベースドライバを追加する必要があります。詳細は、IDE オンラインヘルプを参照してください。



7. 「完了」をクリックします。

新しい JDBC データソースノードが作成されます。



注 – JDBC データソースは、JDBC SQL モデル (表およびストアドプロシージャ) を作成するとき、設計時にのみ必要となります。JDBC SQL モデルウィザードは、作成済みデータソースの選択肢を示します。

JDBC データソースは、実行時環境では必要とされません。直接 JDBC URL を使ってデータベースに接続する場合を除き、適切な JNDI 設定で実行時コンテナを構成する必要があります。

Tomcat (および他の非 JNDI コンテナ) SQL 接続の準備

注 – Sun Java System Application Server を使ってチュートリアルアプリケーションを実行している場合は、JNDI がサポートされているため、この手順をとばすことができます。

組み込み型の Tomcat エンジンを使用している場合、または JNDI をサポートしない別のサーブレットコンテナでチュートリアルアプリケーションを実行している場合は、アプリケーションのアプリケーションサーブレットのベースクラス (JatoTutorialAppServletBase) にいくつかの小さな変更を加える必要があります。

---

1. 「アプリケーションクラス」フォルダを展開します。
2. 「`jatotutorial`」パッケージフォルダを展開します。
3. 「`JatoTutorialAppServletBase`」クラスをダブルクリックして、これを開きます。

   ここには、イベントに対して行うことができることを説明したコメントの入ったイベントコードが、コメントとして多数含まれています。これらはこのチュートリアルアプリケーションでは不要なため、無視してください。

   以下を実行する静的初期化子を追加する必要があります。

   a. JDNI 検索を使用しないように、Web アプリケーションフレームワークに指示します。

   b. PointBase JDBC ドライバを読み込みます。

   c. JDBC データソース (jdbc/jdbc-pointbase) を、PointBase サンプルデータベースの JDBC 接続 URL にマップします。

   以下のコードサンプルは、JatoTutorialAppServletBase クラスに追加する必要があるコードを示しています。追加する必要があるのは、`static` セグメントのみです。ここでは、JatoTutorialAppServletBase クラスのコードやコメントの多くが省略されています。

```
public class JatoTutorialAppServletBase extends ApplicationServletBase {
    static {
        // JNDI 検索をオフにする (DriverManager の使用をオンにする)
        SQLConnectionManagerBase.setUsingJNDI(false);

        try {
            // PointBase JDBC ドライバを読み込む
            Class.forName("com.pointbase.jdbc.jdbcUniversalDriver");
        } catch (ClassNotFoundException e) {
            // ドライバを使用できない場合は、例外がスローされる
            e.printStackTrace();
        }

        SQLConnectionManagerBase.addDataSourceMapping("jdbc/jdbc-pointbase",
                "jdbc:PointBase://localhost:9092/sample");
    } // static init
}
```

これでアプリケーションは、JNDI を介して接続プールを使用するのではなく、JDBC URL を直接使ってデータベースへの接続を作成するようになります。

重要：Web アプリケーションのパフォーマンスを重視する場合は、本稼働には JNDI を使用してください。

- Sun Java System Application Server でチュートリアルをテストしない場合は、PointBase クライアントライブラリ JAR ファイル (`pbclient42RE.jar`) を `WEB-INF/lib` ディレクトリにコピーする必要があります。

  PointBase クライアントライブラリは、以下のディレクトリに格納されています。

  `<studio-install-dir>/appserver7/pointbase/client_tools/lib/`

- 異なるデータベースを使用する場合は、そのデータベースベンダーのクライアントライブラリを `WEB-INF/lib` ディレクトリまたはサーブレットコンテナの `lib/ext` ディレクトリ (クラスパス内) に配置する必要があります。これを Web アプリケーションの `WEB-INF/lib` ディレクトリに配置してください。
- Sun Java System Application Server の場合は、PointBase ライブラリがすでにそのクラスパスに含まれているため、この作業は必要ありません。

第8章

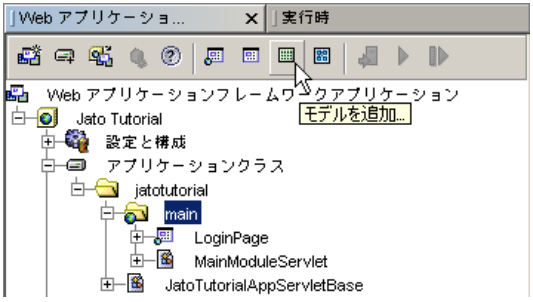
2.2: CustomerModel の作成

この章では、Web アプリケーションフレームワークアプリケーションで RDBMS にアクセスするモデルを作成する方法を説明します。

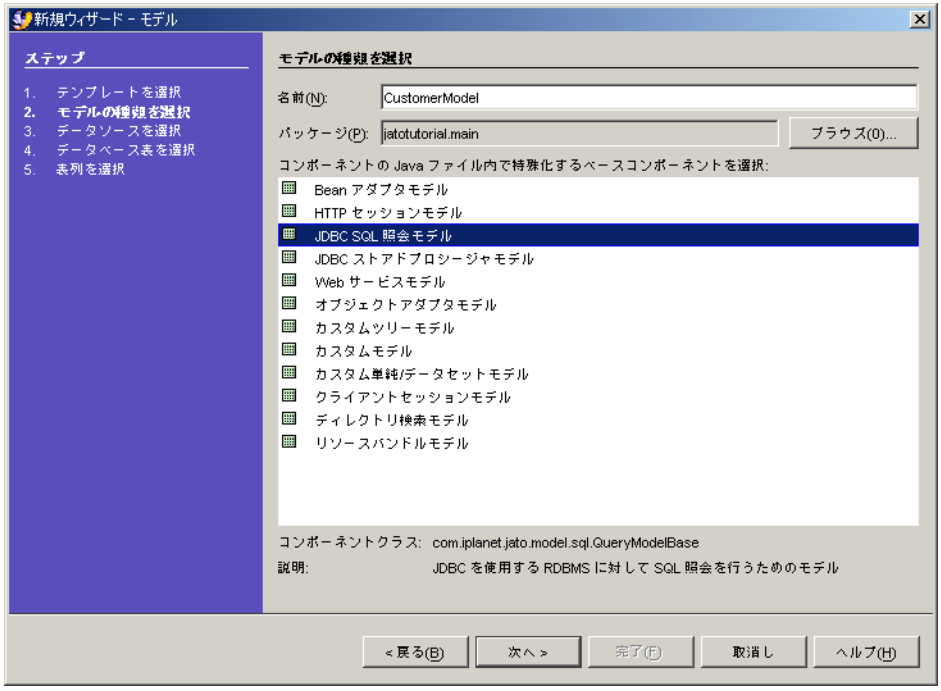
作業 2: CustomerModel の作成

JDBC™ SQL モデルの作成

1. 「main」モジュールフォルダを選択します。
2. Web アプリケーションフレームワークツールバーの「モデルを追加」ボタンをクリックします。



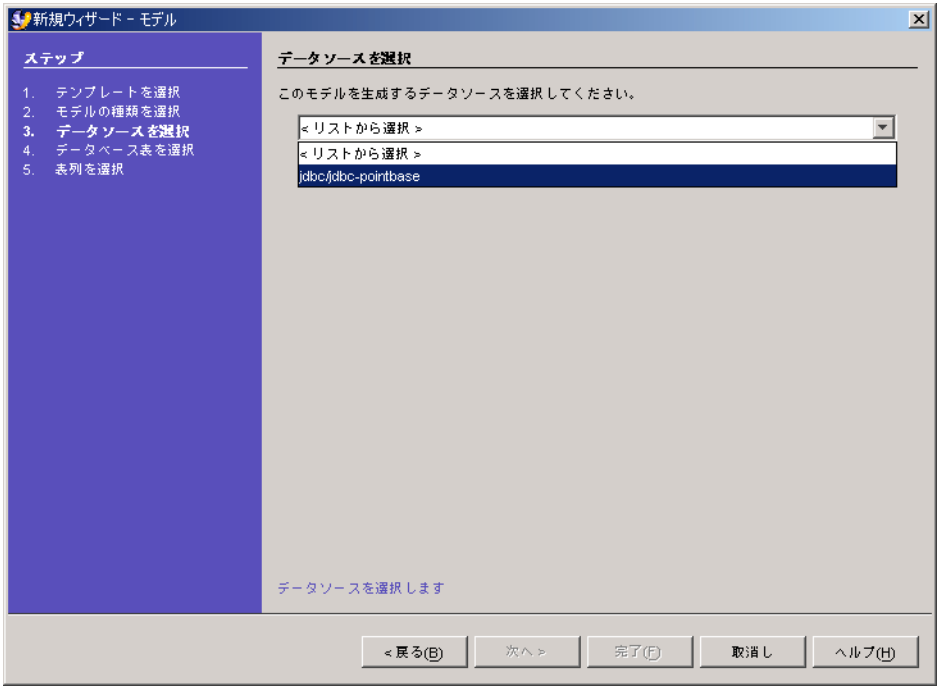
「モデルの種類を選択」パネルが表示されます。



3. 「名前」フィールドに「CustomerModel」と入力します。
4. モデルコンポーネントリストから「JDBC SQL 照会モデル」を選択します。

   Web アプリケーションフレームワークのバージョンやカスタムまたは Sun 以外のコンポーネントライブラリの追加により、表示されるリストが異なる場合があります。
5. 「次へ」をクリックします。

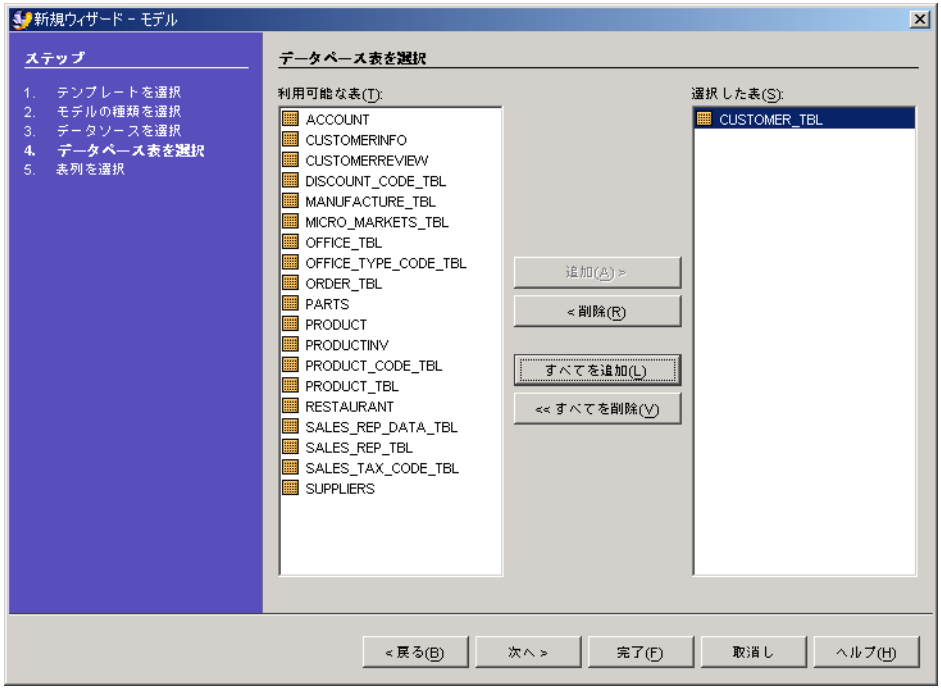
   「データソースを選択」ページが表示されます。



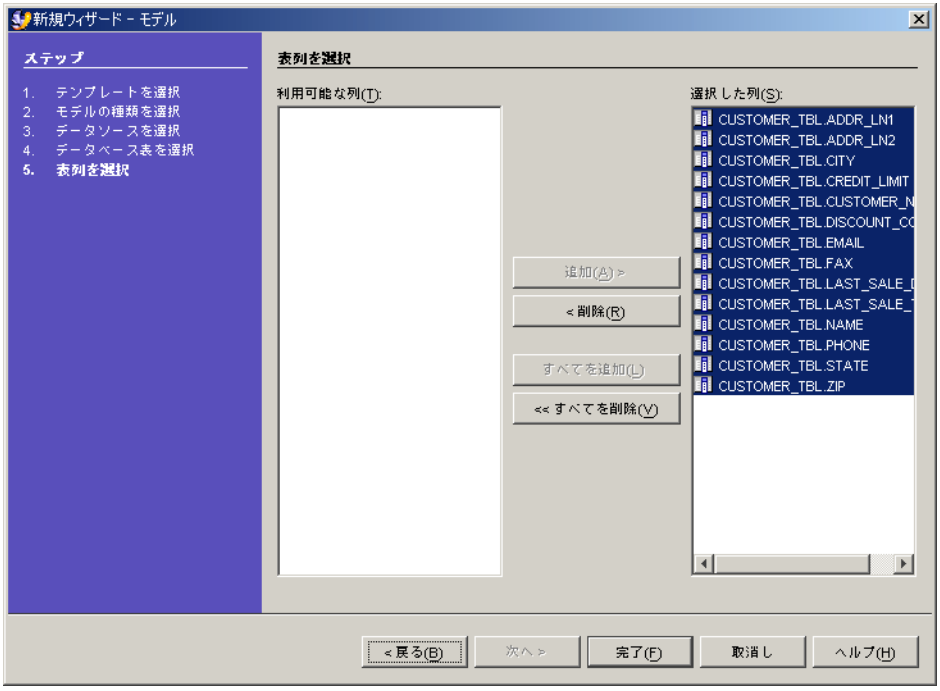
6. コンボボックスから「jdbc/jdbc-pointbase」を選択します。
7. 「次へ」をクリックします。

「データベース表を選択」ページが表示されます。

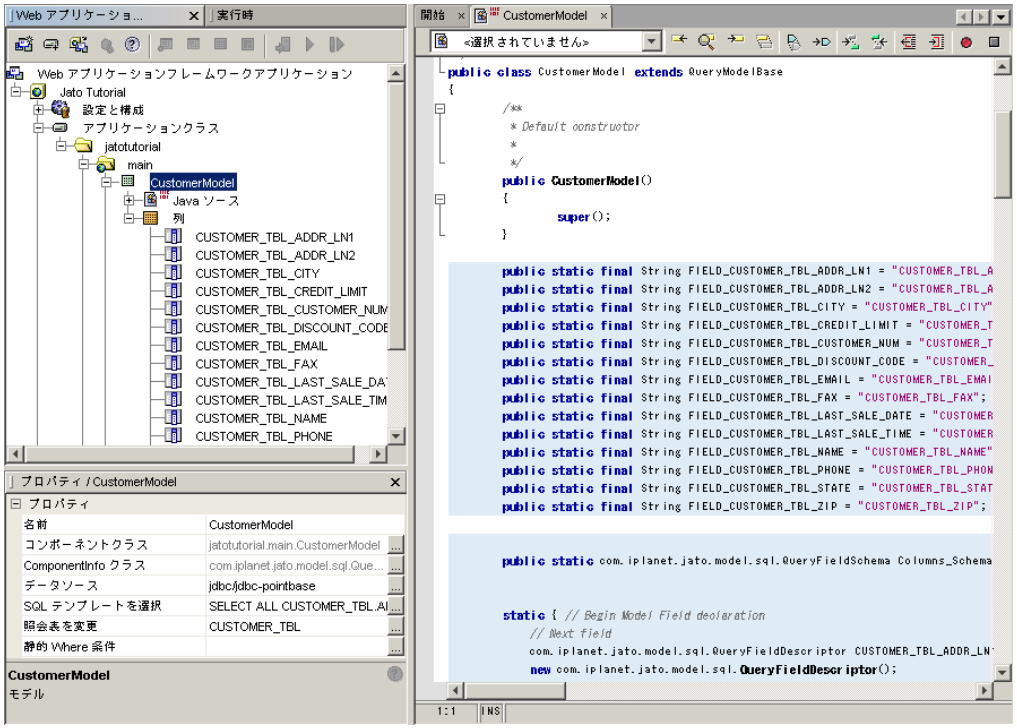
注 – 「次へ」をクリックして「接続」ダイアログが表示される場合は、接続資格の入力が必要になることがあります。(サンプルの PointBase データベースの場合は、ログイン名が「pbpublic」、パスワードが「pbpublic」です。)



8. CUSTOMER_TBL を選択します。
9. 「追加」をクリックします。
10. 「次へ」をクリックします。
    「表列を選択」ページが表示されます。



11. 「すべてを追加」をクリックして、すべての列をモデルに含めます。
12. 「完了」をクリックして、モデルを作成します。
    CustomerModel オブジェクトが main モジュールに作成されます。



13. CustomerModel を展開して、すべての列を表示します。

14. 「CustomerModel」フォルダをダブルクリックして、CustomerModel Java クラスのコードを表示します。

モデルのキーフィールドの指定

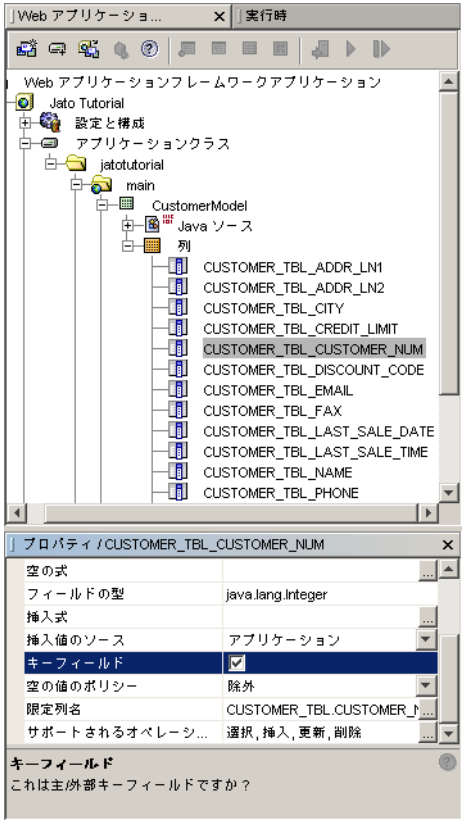
注 – PointBase データベーススキーマメタデータには特殊な型のキーフィールドインジケータがあり、モデルウィザードはキーフィールド CUSTOMER_TBL_CUSTOMER_NUM を正しく検出しません。したがって、キーフィールドは手動で設定する必要があります。

これは、データベーススキーマオブジェクトからデータソースを作成した場合や、Oracle などの非 PointBase データベースでは、問題になりません。

---

1. CustomerModel の「列」ノードの下で、CUSTOMER_TBL_CUSTOMER_NUM モデルフィールドを選択します。
2. プロパティシートで、「モデルフィールドプロパティ」タブを選択します。

   「プロパティ」タブが表示されない場合は、「ウィンドウ」>「プロパティ」メニューオプションをクリックするか、キーフィールド列を右クリックして「プロパティ」アクションを選択します。
3. 「キーフィールド」プロパティの値を「false」から「true」に変更します。



非 JNDI 対応コンテナの接続コードの追加

JNDI データソースをサポートしないサーブレットコンテナでは、JDBC ドライバを明示的に使用することで対応できます。

注 – このチュートリアルの 2.1 (作業 1: SQL データベースへのアクセス) で、JNDI をサポートしないサーブレットコンテナでアプリケーションをテストする場合は、JNDI を使用不可にして、アプリケーションサーブレットクラス (JatoTutorialAppServlet) における PointBase JDBC ドライバの明示的使用を宣言してあります。

JNDI をサポートしないサーブレットコンテナでテストを行う場合は、モデルの実行前に適切なデータベース接続を作成できるように、接続ユーザー名とパスワードをモデルに明示的に設定する必要があります。

注 – 本稼働環境では、JNDI 接続を使用する必要があります。

CustomerModel のコンストラクタに、以下のコードを追加します。

```
public CustomerModel() {
    super();
    setDefaultConnectionUser("pbpublic");
    setDefaultConnectionPassword("pbpublic");
}
```

実際のアプリケーションでは、上記のようにデータソースのユーザー名とパスワードを提供するのは推奨されませんが、このチュートリアルでは有用です。セキュリティー保護を厳重にして、堅牢な実装にするには、ユーザー名とパスワードの取得や提供に細心の注意を払ってください。JNDI の方法を使用する場合はこのコードは不要で、このログイン情報はアプリケーションサーバーの構成された JNDI 接続によって提供されます。

第9章

2.3: 顧客ページの作成

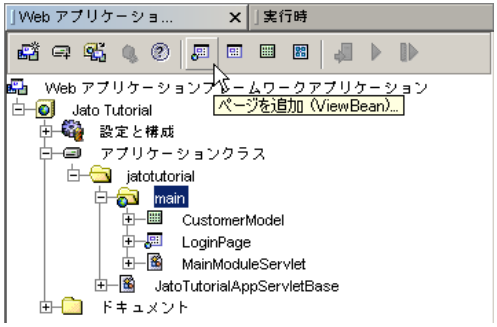
この章では、リレーショナルデータベースにアクセスするモデルのデータを表示するページを Web アプリケーションフレームワークで作成する方法を説明します。

作業 3: 顧客ページの作成

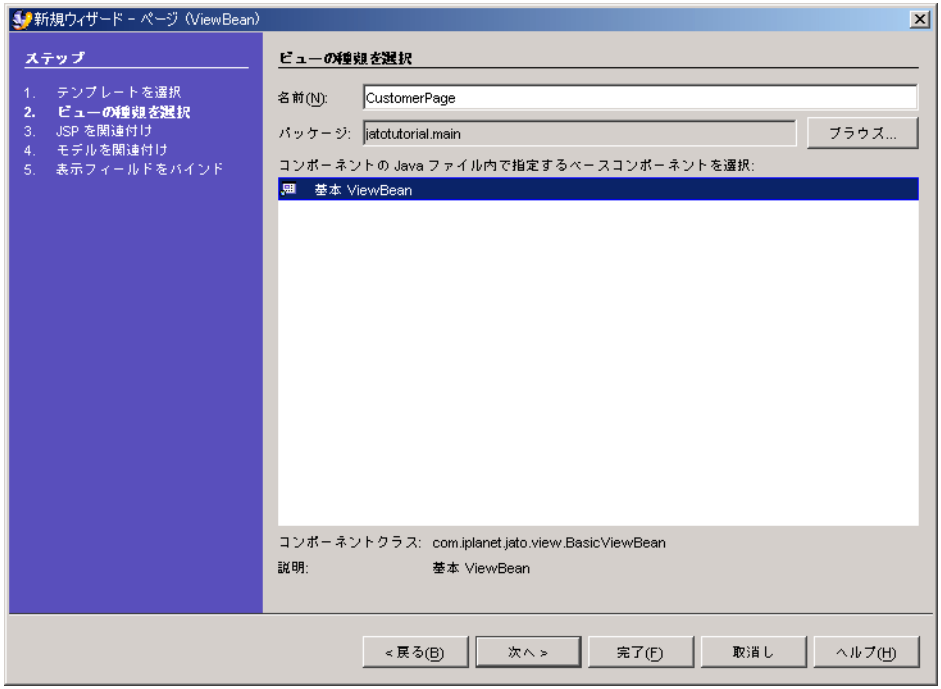
これから、アプリケーションの 2 番目のページを作成します。ただし、このページはモデルにバインドされます。このバインドプロセスにより、モデルのフィールドに格納されているデータを表示する表示フィールドがページ上に自動的に作成されます。

ViewBean の追加

1. 「main」モジュールを選択します。
2. Web アプリケーションフレームワークツールバーの「ページを追加 (ViewBean)」ボタンをクリックします。

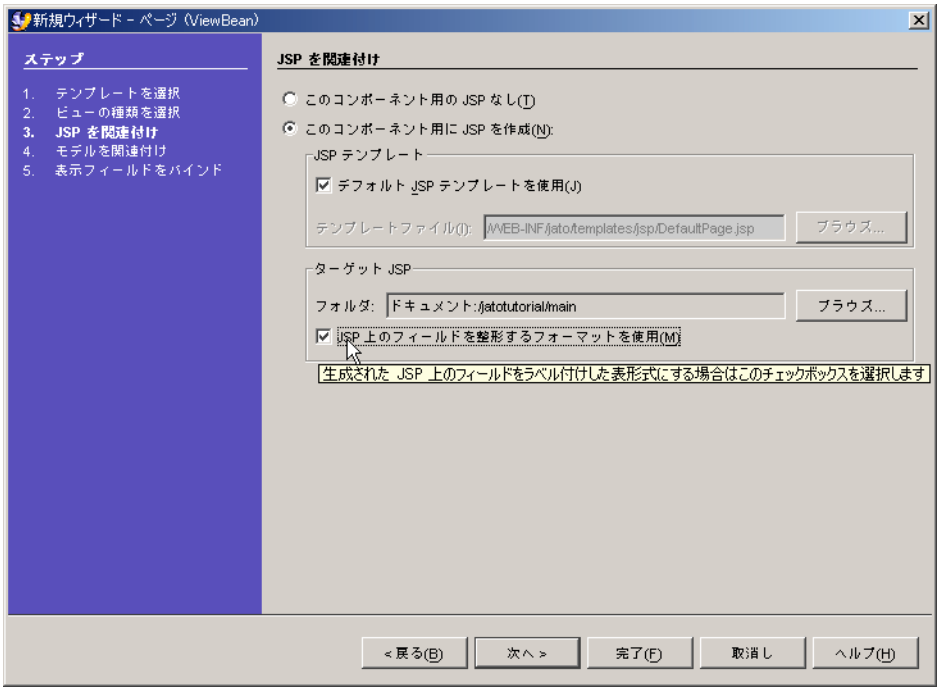


「ビューの種類を選択」パネルが表示されます。



3. 「名前」フィールドに CustomerPage と入力して、<デフォルト名> を置き換えます。
4. 「基本 ViewBean」を選択して、ViewBean タイプのページコンポーネントを作成します。
5. 「次へ」をクリックします。

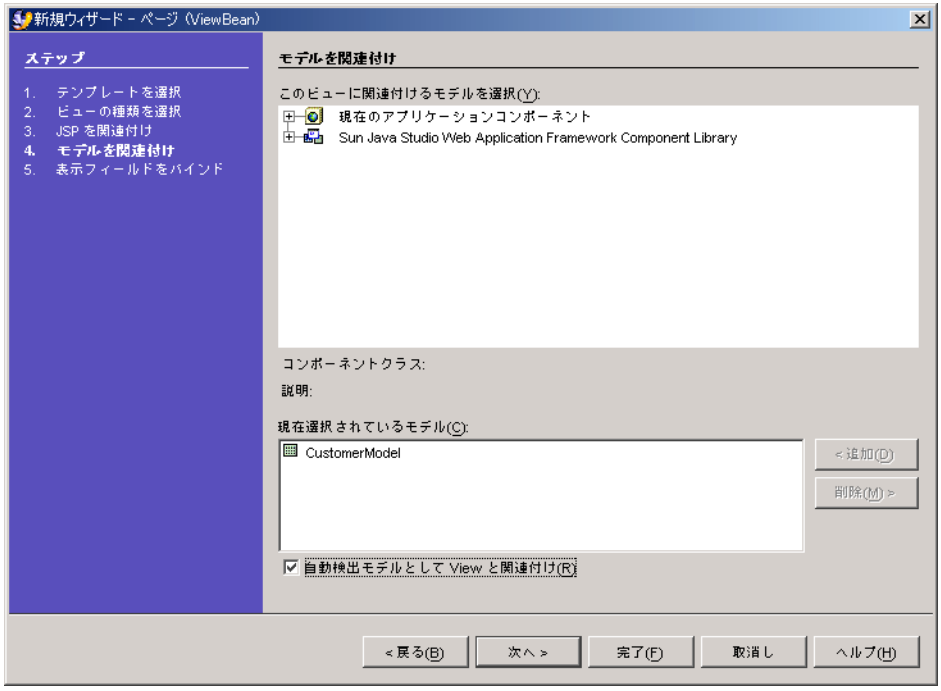
   「JSP を関連付け」パネルが表示されます。



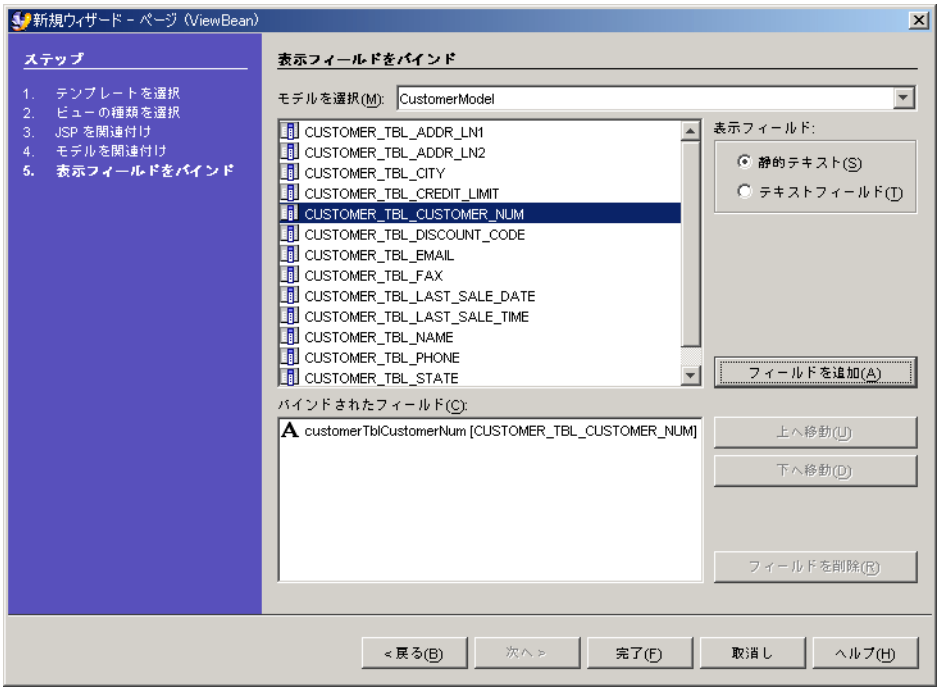
6. 「JSP 上のフィールドを整形するフォーマットを使用」チェックボックスをクリックして、基本的な書式を設定します。

7. 「次へ」をクリックします。

   「モデルを関連付け」パネルが表示されます。



8. 「現在のアプリケーションコンポーネント」を展開して、「jatotutorial」>「main」を表示します。
9. 「CustomerModel」を選択します。
10. 「追加」をクリックします。
11. 「自動検出モデルとしてビューと関連付け」チェックボックスを選択します。
    これを選択すると、このモデルで自動検出モデル機能が使用されるようになります。
12. 「次へ」をクリックします。
    「表示フィールドをバインド」パネルが表示されます。



追加する必要があるのは、3 つのフィールドだけです。

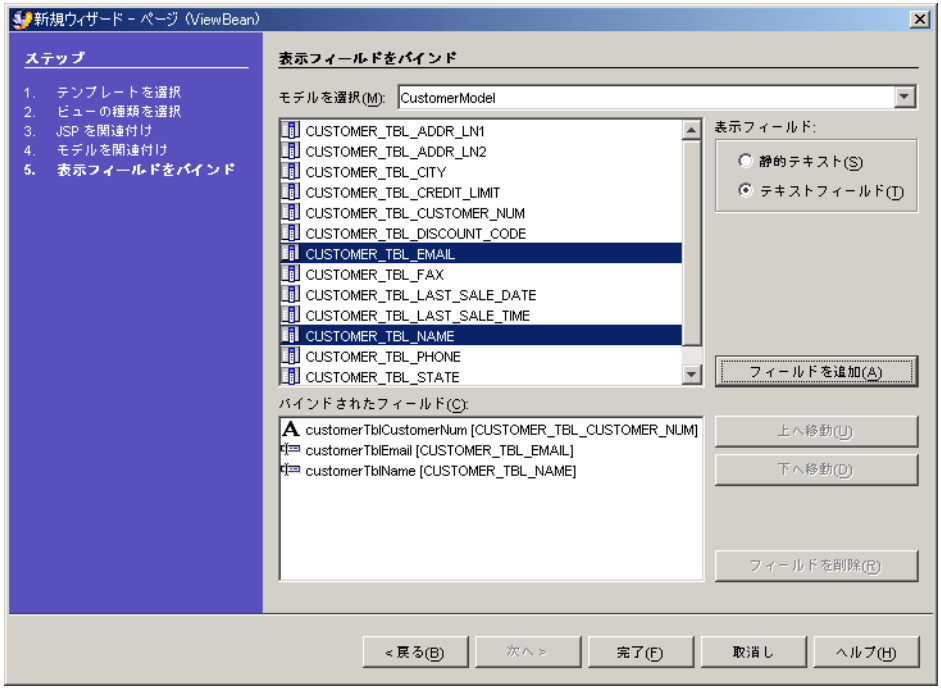
13. 最初のフィールドを追加します。

    a. 「CUSTOMER_TBL_CUSTOMER_NUM」フィールドを選択します。

       デフォルトの「静的テキスト」をそのまま使用します。

    b. 「フィールドを追加」をクリックします。

       「CUSTOMER_TBL_CUSTOMER_NUM」フィールドが「バインドされたフィールド」リストボックスに追加されます。



14. 2 番目と 3 番目のフィールドを同時に追加します。

    a. 「CUSTOMER_TBL_EMAIL」および「CUSTOMER_TBL_NAME」フィールドを選択します (連続していないフィールドを複数選択するには、Ctrl キーを押しながら選択します)。

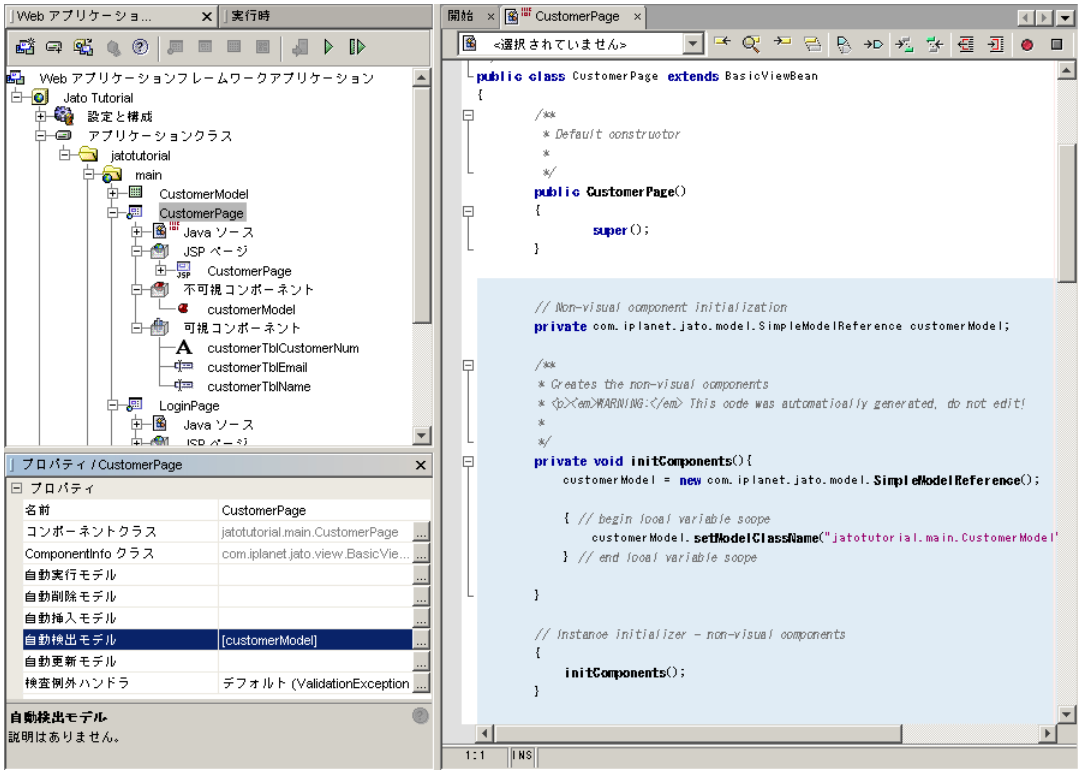
    b. 「テキストフィールド」を選択します。

    c. 「フィールドを追加」をクリックします。

    「CUSTOMER_TBL_EMAIL」および「CUSTOMER_TBL_NAME」フィールドが「バインドされたフィールド」リストボックスに追加されます。

15. 「完了」をクリックします。

    これで ViewBean が作成されます。


Web アプリケーショ...
実行時
Web アプリケーションフレームワークアプリケーション
Jato Tutorial
設定と構成
アプリケーションクラス
jatotutorial
main
CustomerModel
CustomerPage
Java ソース
JSP ページ
CustomerPage
不可視コンポーネント
customerModel
可視コンポーネント
customerTblCustomerNum
customerTblEmail
customerTblName
LoginPage
Java ソース
プロパティ / CustomerPage
プロパティ
名前
CustomerPage
コンポーネントクラス
jatotutorial.main.CustomerPage
ComponentInfo クラス
com.iplanet.jato.view.BasicVie...
自動実行モデル
自動削除モデル
自動挿入モデル
自動検出モデル
[customerModel]
自動更新モデル
検査例外ハンドラ
デフォルト (ValidationException
自動検出モデル
説明はありません。
開始
CustomerPage
<選択されていません>
public class CustomerPage extends BasicViewBean
{
/**
* Default constructor
*
*/
public CustomerPage()
{
super();
}
// Non-visual component initialization
private com.iplanet.jato.model.SimpleModelReference customerModel;
/**
* Creates the non-visual components
* <p>WARNING: This code was automatically generated, do not edit!
*
*/
private void initComponents(){
customerModel = new com.iplanet.jato.model.SimpleModelReference();
{ // begin local variable scope
customerModel.setModelClassName("jatotutorial.main.CustomerModel"
} // end local variable scope
}
// Instance initializer - non-visual components
{
initComponents();
}
1:1
INS

16. 「CustomerPage」をダブルクリックします。

右側のパネルにコードが表示されます。

「CustomerPage」の全サブノードを展開して、ウィザードにより自動的に生成された「JSP ページ」、「可視コンポーネント」、および「不可視コンポーネント」を表示します。

注 – LoginPage と同様に、CustomerPage の JSP が「ドキュメント」フォルダ (`/jatotutorial/main`) に追加されて、この ViewBean の JSP フォルダの下にその JSP へのリンクが作成されます。

CustomerModel のフィールドにバインドするよう指示したため、3 つの可視コンポーネントが作成、表示されます。これにより、データが顧客ページに自動的に表示されて、これらのフィールドに対する変更がモデルに自動的に対応付けられます。変更がモデルに対応付けられたら、データベースの更新をモデルに実行させることができます。SQL の生成と接続の作成は、すべて Web アプリケーションフレームワークによって処理されます。

JDBC API に対して直接作業を行う必要がある場合は、Application Framework の全機能を使用する必要はなく、すべての JDBC 作業を自分で行うことができます。つまり、Web アプリケーションフレームワークで必要な機能を選択できますが、たいていの場合は Web アプリケーションフレームワークはまさに必要なものを実装します。

また、「不可視コンポーネント」ノードの下に、CustomerModel クラスへの「参照」のエントリが表示されます。

ボタンコンポーネントの追加

1. CustomerPage にボタンを追加します。

   コンポーネントパレットを使って、ログインページのフィールドを追加したときのようにボタンを追加します。

2. コンポーネントパレットが表示されていない場合は、以下のメニューオプションを選択します。

   「ウィンドウ」>「Web アプリケーションフレームワーク」>「コンポーネントパレット」

   以下の表には、CustomerPage にボタンを追加するために指定する値が示されています。

種類	名前	初期値
ボタン	update	Update

3. ボタンを使用可能にして、顧客レコードを更新します。

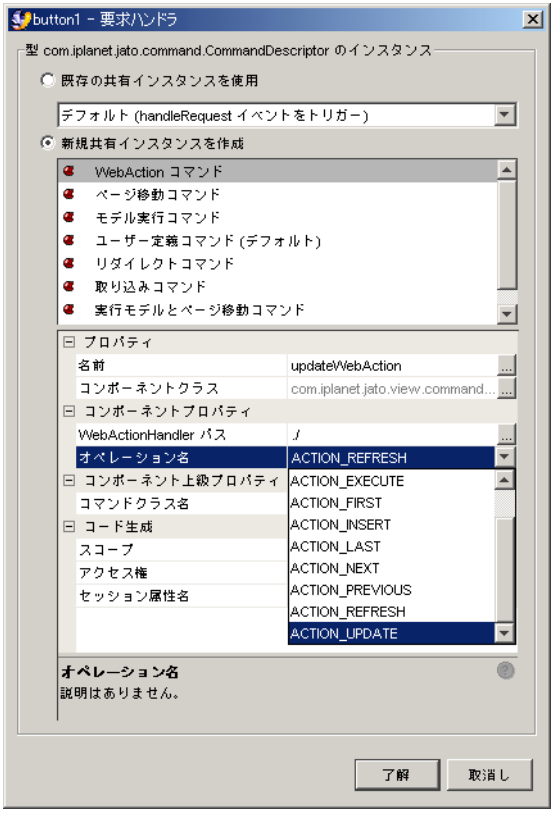
   a. 「update」ボタンフィールドを選択します。

   b. ボタンのプロパティシートで、「要求ハンドラ」プロパティの値領域をクリックします。

      省略符号ボタン (...) が表示されます。

   c. 省略符号ボタン (...) をクリックします。

      コマンド記述子エディタが起動します。



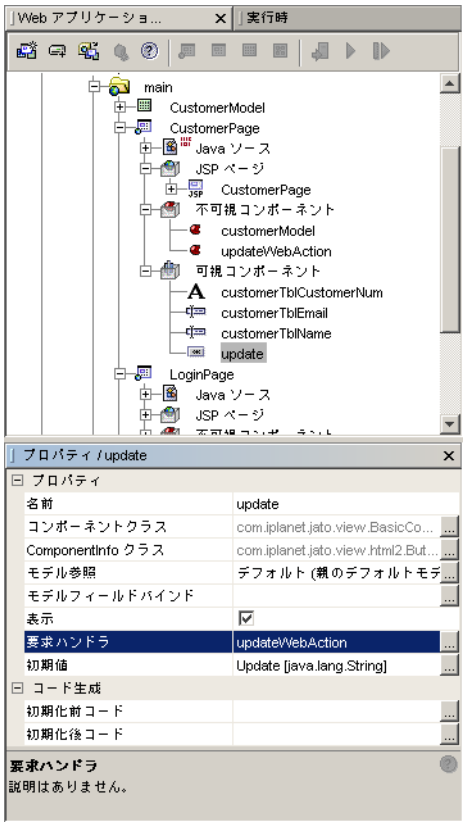
4. 「新規共有インスタンスを作成」を選択します。
5. 一覧から「WebAction コマンド」を選択します。
6. 「プロパティ」セクションで、「updateWebAction」に名前を変更します。
7. エディタ下部の「コンポーネントプロパティ」タブを選択します。
8. 「オペレーション名」プロパティに「ACTION_UPDATE」を選択します。

   他の 2 つのプロパティについては、デフォルトを受け入れます。
9. 「了解」をクリックします。

   これで、このプロパティの設定が完了しました。

注 – 「不可視コンポーネント」ノードの下に新しいエントリが追加され、「コマンド記述子」プロパティが設定されます。

次の図は、「update」ボタンのプロパティシートを示しています。「要求ハンドラ」プロパティには、前述の手順で構成した新しいコマンド記述子が指定されています。



モデル自動更新機能の作成

customerModel 参照を、CustomerPage で「自動更新モデル」として追加する必要があります。

これは、ページの「自動更新モデル」プロパティに適切なモデル参照を設定することによって行います。この場合は、モデルの関連付けやフィールドバインドを指定した結果としてウィザードにより作成された customerModel 参照になります。

1. 「CustomerPage」ノードを選択します。
2. 「自動更新モデル」プロパティの値領域をクリックします。

   省略符号ボタン (...) が表示されます。

3. **省略符号ボタン (...) をクリックします。**

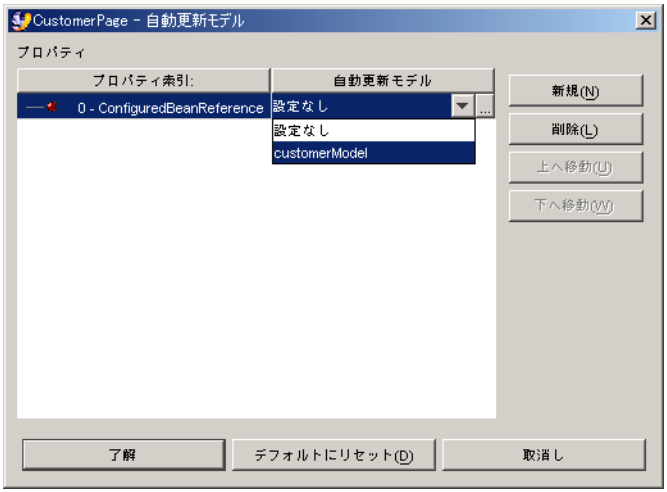
   「自動更新モデル」カスタムエディタが起動します。

   このエディタが表示されたとき、最初は「プロパティ」領域が空白になっている場合は、次の設定を行います。

4. **「新規」をクリックします。**

   これによってエントリを追加できます。

5. **「自動更新モデル」コンボボックスから「customerModel」を選択します。**



6. **「了解」をクリックします。**

   これでプロパティに customerModel エントリが追加されます。

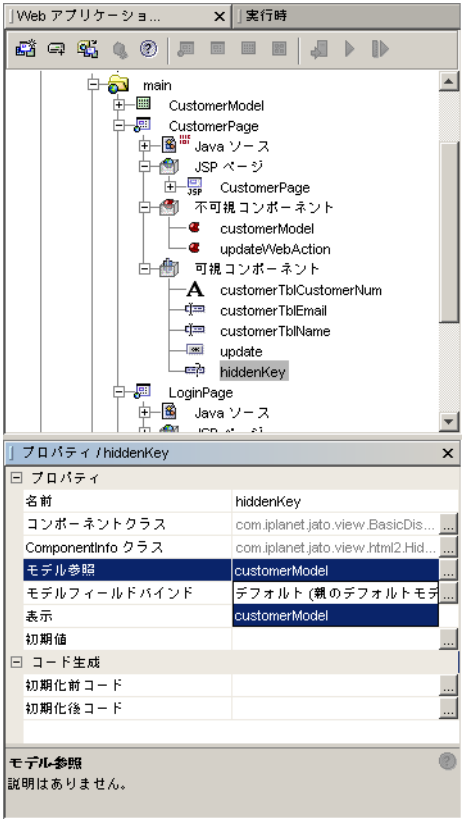
注 – ボタンの Web 更新アクションコマンド記述子とモデル自動取り出しまたは更新を組み合わせて設定すると、顧客ページが表示されたとき、または CustomerPage の「Update」ボタンがクリックされたときに CustomerModel が実行されるようになります。

このモデルで自動実行を宣言する代わりに、同じ処理を実行するコードを記述することもできます。一般的に、このコードは「Update」ボタンの handleUpdateRequest イベントに (ログインページの「Login」ボタン用にコードを実装したのと同様に) 実装します 。

顧客ページへの非表示フィールドの追加

1. 「CustomerPage」ノードを展開します。
2. 「可視コンポーネント」ノードを展開します。
3. 「可視コンポーネント」ノードを選択します。
4. コンポーネントパレットを使用して非表示フィールドを追加します。

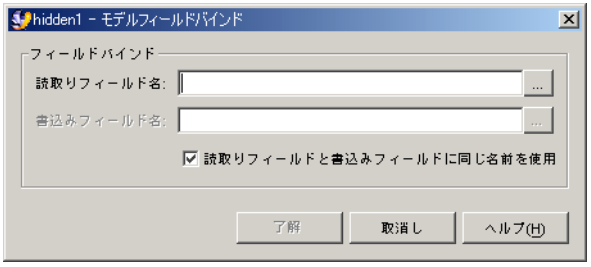
   「CustomerPage」の「可視コンポーネント」ノードに非表示フィールドが追加されます。
5. 非表示フィールドの名前を「hiddenKey」に変更します。
6. 「hiddenKey」フィールドを、「customerTblCustomerNum」静的テキストフィールドがバインドされているのと同じモデルフィールドにバインドします。
7. 「hiddenKey」フィールドを選択します。
8. プロパティシートで、ドロップダウンボックスから「モデル参照」プロパティを選択して、「customerModel」に設定します。



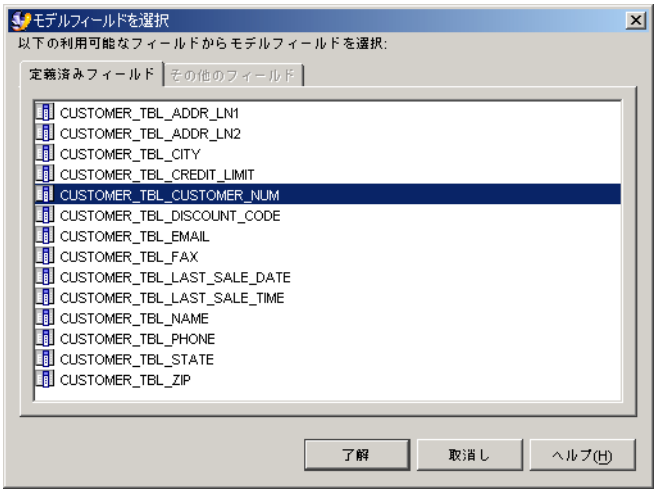
9. 「モデルフィールドバインド」プロパティを設定します。

   a. 「モデルフィールドバインド」プロパティの値領域をクリックします。

   b. 省略符号ボタン (...) をクリックして、「モデルフィールドバインド」プロパティエディタを起動します。



10. 「読取りフィールド名」プロパティの省略符号ボタン (...) をクリックして、「モデルフィールドを選択」エディタを起動します。

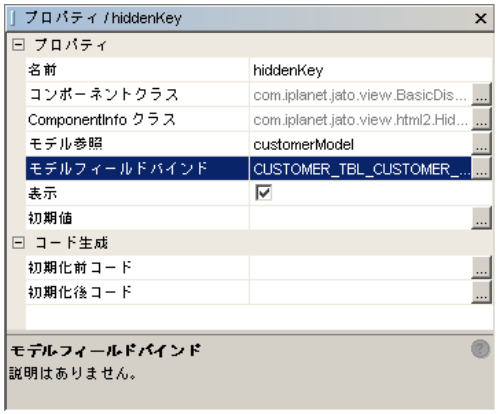


11. 「CUSTOMER_TBL_CUSTOMER_NUM」を選択します。
12. 「了解」をクリックします。

    「CUSTOMER_TBL_CUSTOMER_NUM」モデルフィールドを使って、読取り/書込みフィールドが生成されます。

13. 「了解」をクリックします。

    これで、「hiddenKey」表示フィールドの「モデルフィールドバインド」プロパティの設定は完了です。



静的テキストフィールドは HTML 入力フィールドではないため、「update」ボタンをクリックしても、その値はサーバーに送付されません。また、顧客フィールドはデータベース表のキーフィールドであるため、1 つのデータベース行に更新を制限するために、更新ロジックはこのキー値を必要とします。表の全レコードを更新するのではなく、適切な顧客レコードで更新を実行できるように、この値は他の入力フィールド値とともに送信される必要があります。このためには、フォーム送付時に送信し、「CUSTOMER_TBL_CUSTOMER_NUM」モデルフィールドに対応づける非表示フィールドに顧客番号フィールド値を保持します。

注意 – この作業を行わないと、フォームとともにキーフィールド値が送付されなくなります。この結果、JDBC update 文から WHERE 句が抜けることになり、意図せずして表全体が変更されることになります。

これは、実際の作業でこれを実装するやり方とは異なります。セキュリティ上の理由から、HTML でキーフィールドを入力フィールドとして表示する方法は避けるのが一般的です。

注 – これは Web アプリケーションフレームワークに固有の問題ではなく、Web アプリケーションを実装するためのフレームワーク (または非フレームワーク) にかかわらず、あらゆる Web アプリケーションで対処すべき問題です。

Web アプリケーションフレームワークは、ページセッションという付加価値機能を提供します。これは、このソリューションをより安全に実装するテクニックを提供しますが、これについてはこのチュートリアルでは説明しません。詳細は、JatoSample アプリケーションと『Web アプリケーションフレームワーク 開発ガイド』を参照してください。

JSP の書式設定

1. 「CustomerPage」ノードの下の「JSP ページ」ノードを展開します。
2. 「CustomerPage」JSP をダブルクリックして、エディタウィンドウでこれを開きます。
3. 大文字と小文字に注意して正しいラベルをフィールドに入力します。

   以下に、JSP に最小限の書式を設定した例を示します (関連するコードのみを示します)。以下の一部の HTML ソースコードは、分かりやすくするために太字で示されています。

```
<jato:form name="CustomerPage" method="post">

<table border=0 cellspacing=2 cellpadding=2 width="100%">
<tr>
  <td align=right valign=middle width="20%"><b>Customer #:</b></td>
  <td align=left valign=middle><jato:text name="customerTblCustomerNum"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Email:</b></td>
  <td align=left valign=middle><jato:textField name="customerTblEmail"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Name:</b></td>
  <td align=left valign=middle><jato:textField name="customerTblName"/></td>
</tr>
</table>

<jato:button name="update"/>
<jato:hidden name="hiddenKey"/>
</jato:form>
```

第 10 章

2.4: 顧客ページのテスト実行

この章では、Web アプリケーションフレームワークアプリケーションの実行方法を説明します。

作業 4: 顧客ページのテスト実行

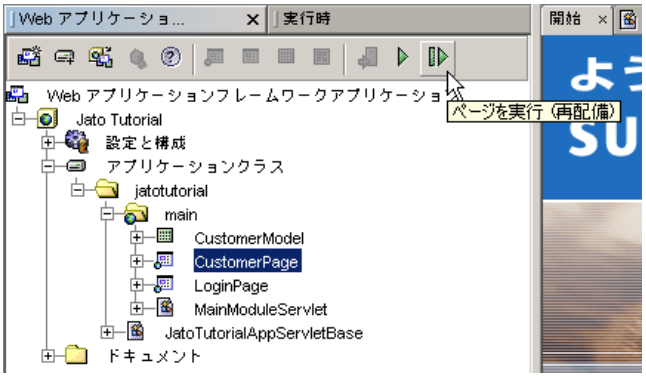
重要：PointBase ネットワークサーバーが動作していることを確認してください。これが動作していない場合は、以下の手順で起動できます。

1. **メニューオプションの「ツール」>「PointBase ネットワークサーバー」>「サーバーを起動」を選択します。**
2. **「アプリケーションクラス」ノードを右クリックします。**
3. **「すべてコンパイル」アクションを選択します。**

   Sun Java System Application Server 上で実行している場合は、変更を加えたときにアプリケーションを配備する必要があります。

4. **「Web アプリケーションフレームワークアプリケーション」ノード (Jato Tutorial) を選択し、Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。**
5. **「CustomerPage」ノードを選択し、「ページを実行 (再配備)」ボタンをクリックします。**

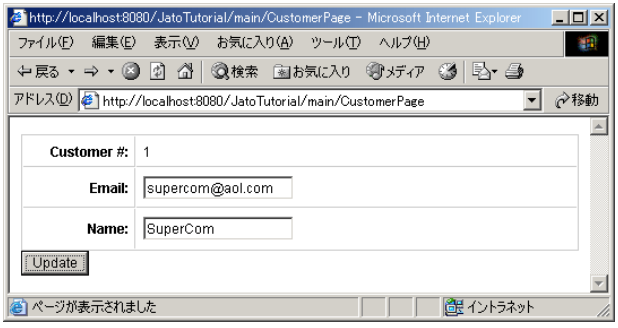
   この実行および再配備オプションを使用すると、サーバーがすべての変更を拾い上げてキャッシュされたリソースを使用しないように、サーバーが再起動します。



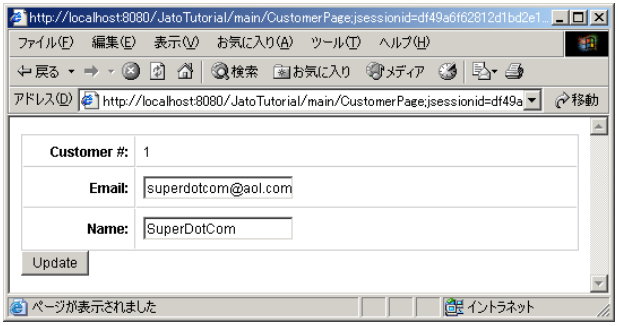
デフォルトのブラウザでアプリケーションが起動します。

顧客更新のテスト

1. フィールドの 1 つまたは両方に変更を加えます。



2. 「Update」をクリックします。



この図では、電子メール名と顧客名が変更されています。

第 11 章

2.5: 顧客ページへのログインページのリンク

この章では、Web アプリケーションフレームワークアプリケーションの LoginPage を CustomerPage にリンクさせて、顧客ページで表示されるデータを顧客のログインに基づいてフィルタリングする方法を説明します。

作業 5: ログインページの顧客ページへのリンク

LoginPage の handleLoginRequest メソッドの編集

`LoginPage.java` ファイルを編集します。

以下のコード例に示されているように `handleLoginRequest()` メソッドのロジックを変更して、ログインが成功した場合は顧客名フィールドに入力された値に対応する顧客データが顧客ページに表示されるようにします。

注 – 以下のコード例では、顧客名フィールドの唯一の正しい値は、顧客表の CustomerID 値でもあります。

したがって、CustomerModel が使用する WHERE 句にログイン ID 値を使用できます。

これによって、CustomerModel が取り出すデータが、適切な CustomerID に確実に対応するようになります。

コードの変更は慎重に行ってください。

以下のコードは実質的に、このイベントで以前に使用されていたコード全体の置き換えです。

デルタに見えるものだけを追加すると、エラーになる可能性があります。現在のコードを削除してから、以下のコードで置き換えることが最善です。

以下に、`handleLoginRequest()` メソッドのロジックを変更するために入力する必要があるコードを示します。

```
public void handleLoginRequest(RequestInvocationEvent event) {
    // 顧客番号を取り出す
    String custNum = getDisplayFieldStringValue(CHILD_CUSTOMER_NUM);

    String theMessage = "";

    // 顧客番号を確認
    // 注: この例ではパスワードはチェックしない
    if (custNum.equalsIgnoreCase("1") || custNum.equals("777") ||
            custNum.equals("410")) {
        // ログインページを返す代わりに、顧客番号に
        // 一致する顧客の顧客ページを表示
        // CustomerModel への参照を取得
        CustomerModel model = (CustomerModel)getModel(CustomerModel.class);

        // ログインに使用された顧客番号を反映して where 条件を変更
        model.clearUserWhereCriteria();
        model.addUserWhereCriterion(
                "CUSTOMER_TBL_CUSTOMER_NUM", new Integer(custNum));

        // 顧客ページの表示
        getViewBean(CustomerPage.class).forwardTo(event.getRequestContext());
    } else {
        theMessage = "Sorry, " + custNum +
                ", your customer number was incorrect!";

        // 出力状態メッセージを設定
        getDisplayField(CHILD_MESSAGE).setValue(theMessage);

        forwardTo();
    }
}
```

第 12 章

2.6: アプリケーションの実行

この章では、別のページを追加してこれを最初のページにリンクさせた 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実行する方法を説明します。

作業 6: アプリケーションの実行

重要：PointBase ネットワークサーバーが動作していることを確認してください。

これが動作していない場合は、以下の手順でこれを IDE で起動できます。

1. **メニューオプションの「ツール」>「PointBase ネットワークサーバー」>「サーバーを起動」を選択します。**

   いくつかのクラスに変更を加えているため、アプリケーションをコンパイルすることが必要です。

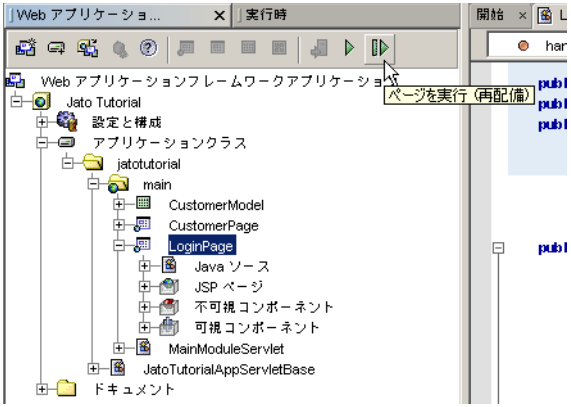
2. **「アプリケーションクラス」ノードを右クリックして、「すべてコンパイル」アクションを選択します。**

   Sun Java System Application Server 上で実行している場合は、変更を加えたときにアプリケーションを配備する必要があります。

3. **「Web アプリケーションフレームワークアプリケーション」ノード (Jato Tutorial) を選択し、Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。**

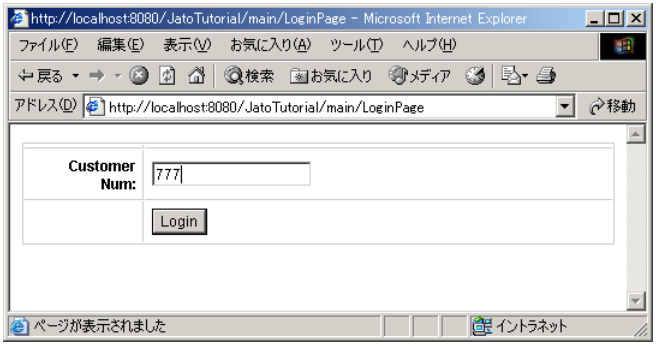
4. **「LoginPage」ノードを選択し、「ページを実行 (再配備)」ボタンをクリックします。**

   この実行および再配備オプションを使用すると、サーバーがすべての変更を拾い上げてキャッシュされたリソースを使用しないように、サーバーが再起動します。



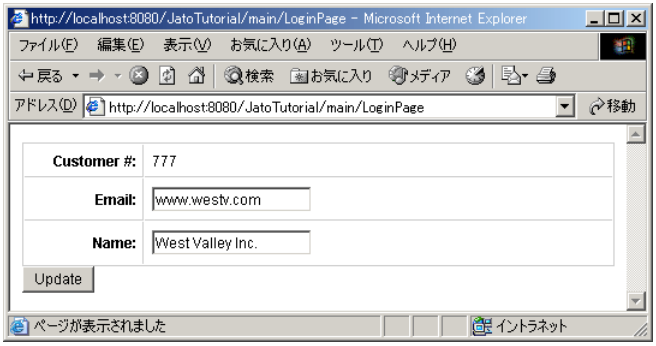
デフォルトのブラウザでアプリケーションが起動します。

5. 有効な顧客番号 (1、777、または 410) を入力します。



6. 「Login」をクリックします。

ログインに使用した顧客番号に対応する顧客のデータが、顧客ページに表示されます。



第 13 章

3.1: コマンドコンポーネントの作成

この章では、同じアプリケーション内の多数のコマンドフィールド (ボタンおよび HREF コンポーネント) で再使用できるコマンドコンポーネントを作成する方法を説明します。これは、その親ページやページレットクラス内でコマンドフィールドの要求処理イベントに要求処理コードを実装するための代替テクニックです (例 : LoginPage の handleLoginRequest)。

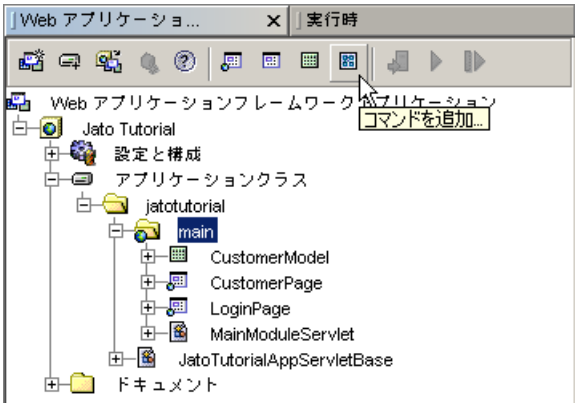
コマンドは、コードの再使用という点で、非常な有用性と柔軟性を提供します。com.iplanet.jato.command.Command インタフェースを実装するだけで、任意の Java クラスをコマンドコンポーネントにできます。このチュートリアルでは、Web アプリケーションフレームワークコマンドウィザードを使って新規 Command クラスを作成し、「Login」ボタンの要求ハンドライベントに代わるログイン/ログアウトコマンドを作成します。このコマンドコンポーネントは、必要に応じて他のページやページレットのコマンドフィールドで再利用できます。

作業 1: コマンドコンポーネントの作成

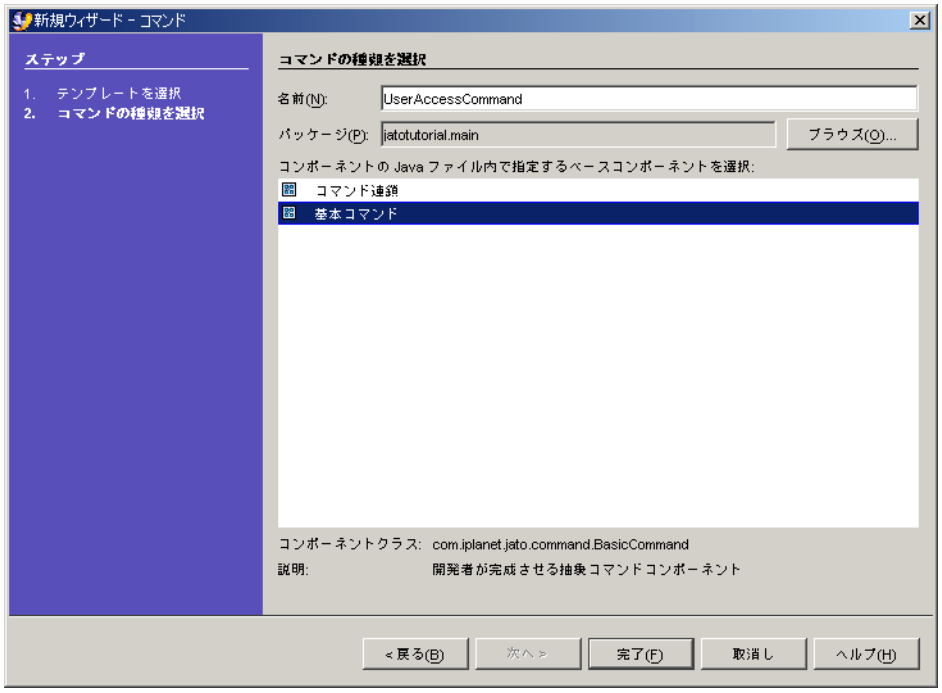
UserAccessCommand コンポーネントの作成

Web アプリケーションフレームワークコマンドウィザードを使って、コマンドコンポーネントを作成します。

1. 「main」モジュールフォルダを選択して、Web アプリケーションフレームワークツールバーの「コマンドを追加...」ボタンをクリックします。

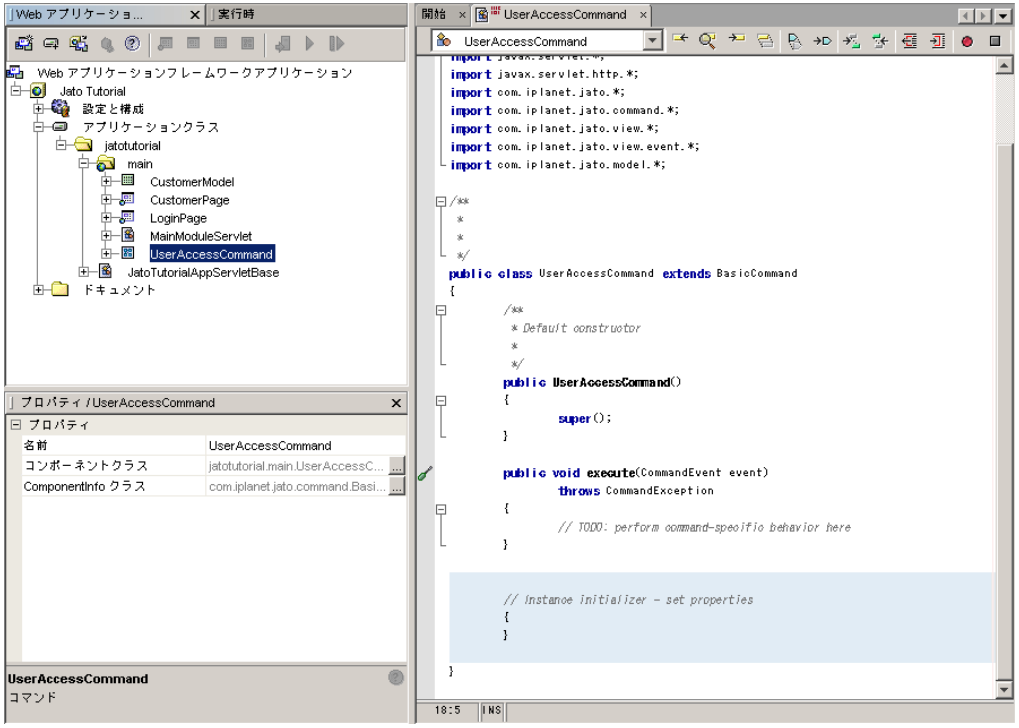


「コマンドの種類調整を選択」パネルが表示されます。



2. 「名前」テキストボックスに、「UserAccessCommand」と入力します。
3. 「基本コマンド」を選択します。
4. 「完了」をクリックします。

UserAccessCommand コンポーネントがアプリケーションに追加されます。



「UserAccessCommand」をダブルクリックして、このコンポーネントの Java ソースコードをエディタウィンドウで開きます。

BasicCommand を拡張するのは、非常に単純なクラスです。BasicCommand は、以下の 1 つのメソッドのみを宣言するコマンドインタフェースを実装します。

```
public void execute(CommandEvent) throws CommmandException
```

このコマンドは、現在のところは何も実行しません。必要な作業、つまり CommandEvent パラメータを通して渡される「オペレーション名」に基づいてユーザーログインまたはログアウトを実行させるためのコードを、この execute メソッドに追加する必要があります。オペレーション名は、開発者が自由に指定できます。以下の作業では、カスタムのオペレーション名を渡して評価する方法について説明します。

execute メソッドへのコードの追加

この作業では、多少のコードを記述する必要があるだけです。大量のコードのように見えるかも知れませんが、この多くは LoginPage の handleLoginRequest イベントの再実装です。これによって、ボタンイベントが必要とするものに置き代わります。

UserAccessCommand クラスの execute メソッドに以下のコードを追加してください。

```
public void execute(CommandEvent event) throws CommandException {
 // RequestContext を取得
 RequestContext requestContext = event.getRequestContext();

 // J2EE HttpSession を取得
 javax.servlet.http.HttpSession session =
 requestContext.getRequest().getSession();

 // commandfield オブジェクト (ボタン/href) により
 // 渡されたオペレーション名を取得
 String opName = event.getOperationName();

 // LoginPage を取得
 LoginPage loginVB = (LoginPage)requestContext
 .getViewBeanManager().getViewBean(LoginPage.class);

 // ユーザーログインを実行
 if (opName.equals("login")) {
 // 入力された顧客番号を取得
 int custNum = loginVB.getDisplayFieldIntValue(
 LoginPage.CHILD_CUSTOMER_NUM);

 // 顧客モデルを取得
 CustomerModel customerModel = (CustomerModel)requestContext
 .getModelManager().getModel(CustomerModel.class);

 // 顧客番号を条件として使って CustomerModel を実行して
 // データベースにユーザーが存在するかどうかをチェック
 customerModel.clearUserWhereCriteria();
 customerModel.addUserWhereCriterion(
 "CUSTOMER_TBL_CUSTOMER_NUM", new Integer(custNum));
 try {
 customerModel.executeSelect(null);
 } catch (ModelControlException e) {
 Log.log("Exception caught in UserAccessCommand.execute(): "
 + e.toString());
 } catch (java.sql.SQLException e) {
 Log.log("Exception caught in UserAccessCommand.execute(): "
```

```
+ e.toString());
}

// 有効な顧客番号が入力された場合
if (customerModel.getNumRows() == 1) {
// 顧客ページを表示
requestContext.getViewBeanManager().getViewBean(
CustomerPage.class).forwardTo(requestContext);

// 顧客番号を HttpSession 属性に設定して
// 後で要求された場合に使用できるようにする
session.setAttribute("hsaCustNum", new Integer(custNum));
}
// 無効な顧客番号が入力された場合
else {
String msg = "Sorry, " + custNum +
" is not a valid customer number.";

// 出力状態メッセージを設定
loginVB.getDisplayField(
LoginPage.CHILD_MESSAGE).setValue(msg);

// ログインページの表示
loginVB.forwardTo(requestContext);
}
} // opName = login の場合

// ユーザーログアウトを実行
else if (opName.equals("logout")) {
// ログアウトメッセージに使用する顧客番号をセッションから取得
String hsaCustNum = session.getAttribute("hsaCustNum").toString();

// 状態メッセージの値を設定
String msg = "Customer " + hsaCustNum +
", you have logged out successfully.";

// ユーザーの HttpSession を無効にする
session.invalidate();

// ログアウトメッセージを設定してログインページを表示
loginVB.getDisplayField(LoginPage.CHILD_MESSAGE).setValue(msg);

// ログインページの表示
loginVB.forwardTo(requestContext);
} // あるいは opName = logout の場合

// 未知のオペレーション名
else {
throw new CommandException(
```

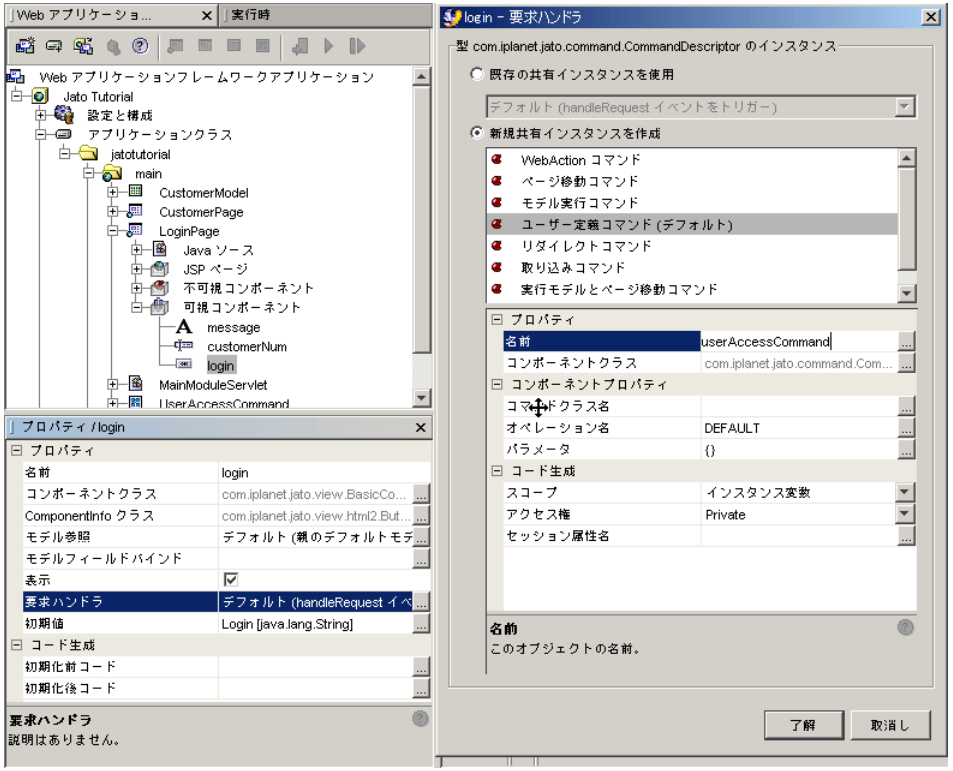
```
  "Unknown UserAccessCommand operation name: " + opName);
  }
}
```

このコードをテスト実行する前に、これを使用するようコマンドフィールド (ボタンまたは HREF) を構成する必要があります。

ボタンのコマンド記述子の構成

ボタンの「コマンド記述子」プロパティ経由で UserAccessCommand コンポーネントを使用するように「login」ボタンを設定します。これは、HREFS のついても同様に機能します。

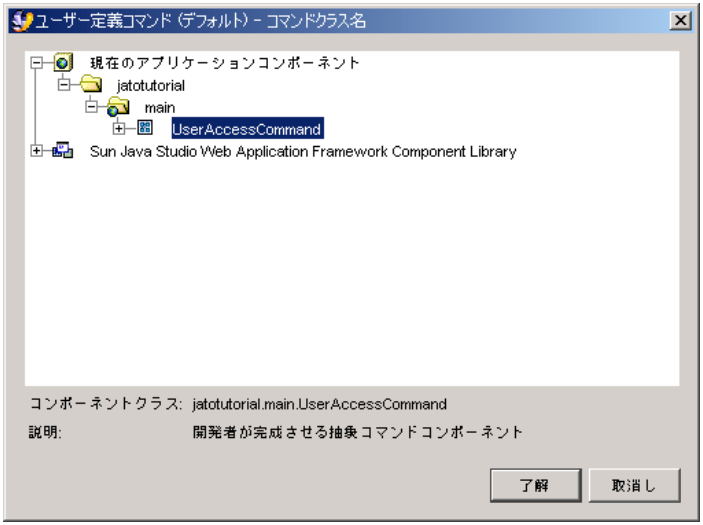
1. 「LoginPage」ノードを展開して、「可視コンポーネント」ノードを展開します。
2. 「可視コンポーネント」の下の「login」ボタンを選択します。



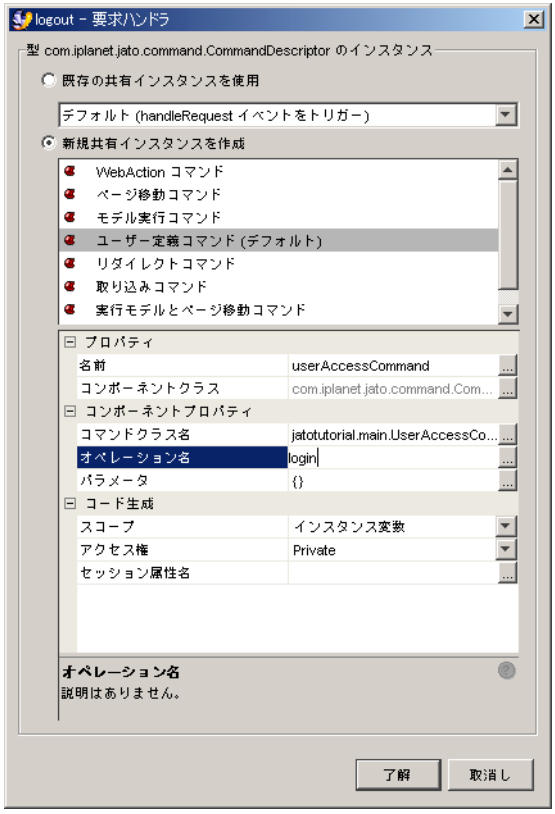
3. 「要求ハンドラ」プロパティの省略符号ボタン (...) をクリックします。

   これによってコマンド記述子エディタが表示されます。

4. 「新規共有インスタンスを作成」ラジオボタンの選択リストから、「ユーザー定義コマンド (デフォルト)」を選択します。
5. 「名前」プロパティを userAccessCommand に変更します。
6. エディタ下部の「コンポーネントプロパティ」タブを選択します。
7. 「コマンドクラス名」プロパティの省略符号ボタン (...) をクリックします。
   「コマンドクラス名」選択のダイアログが表示されます。
8. 「現在のアプリケーションコンポーネント」ノード、「jatoturial」、「main」の順に展開します。
9. 「UserAccessCommand」コマンドコンポーネントを選択します。
10. 「了解」をクリックします。



11. オペレーション名を「DEFAULT」から「login」に変更します。
    コードで UserAccessCommand クラスの execute メソッドに実装したものを思い出してください。このコードには、オペレーション名として「login」または「logout」のいずれかを期待する if/else ブロックがあります。これらは大文字と小文字を区別するため、正しく設定する必要があります。正しく設定しないと、このコマンドのテスト実行時に CommandException (未知のオペレーション名) を受信します。



12. 「了解」をクリックして、「login」ボタンのコマンド記述子プロパティの設定を完了します。

    これで、ログインページを実行して「Login」ボタンをクリックすると、LoginPage の handleLoginRequest イベントのコードの代わりに、UserAccessCommand コンポーネントが要求を処理するようになります。

    handleLoginRequest イベントのコードはそのままにしておくことができます。これは、コマンドコンポーネントの代わりに要求ハンドライベントを使用するように「login」ボタンを再構成しない限り、このコードが呼び出されないためです。

    Web アプリケーションフレームワークは、最初にコマンドフィールドにコマンド記述子がないか調べます。コマンド記述子が実装されていない場合は、handle<CommandField>Request メソッドを呼び出そうとします。メソッドが実装されていない場合は、「要求ハンドラが見つかりません」という例外が表示されます。

第 14 章

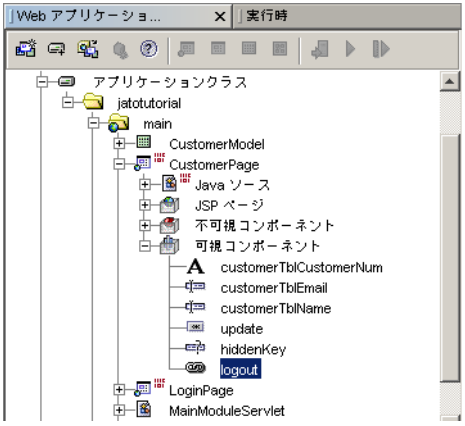
3.2: 顧客ページへのログアウトリンクの追加

この章では、コマンドコンポーネントを使用する HREF をページに追加する方法を説明します。

作業 2: 顧客ページへの HREF の追加

1. 「CustomerPage」ノードを選択します。
2. コンポーネントパレットを使って、ハイパーリンク (HREF) コンポーネントを追加します。

   CustomerPage の「可視コンポーネント」ノードに、HREF コマンドフィールドが追加されます。
3. HREF の名前を「logout」に変更します。

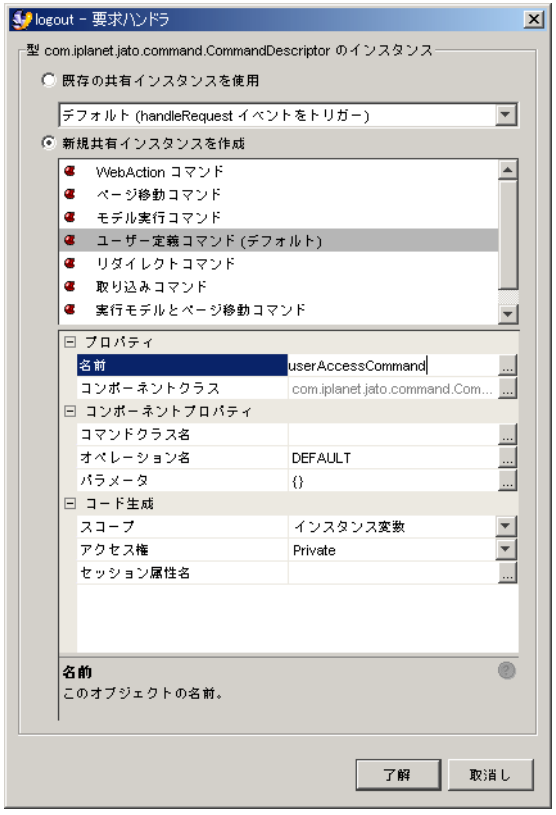


HREF のコマンド記述子の構成

ボタンのコマンド記述子プロパティを介して UserAccessCommand コンポーネントを使用するように、logout HREF を構成します。これは、「オペレーション名」が「login」ではなく「logout」になることを除き、前の作業のボタンコマンド記述子の構成とまったく同じです。

1. 「CustomerPage」の「可視コンポーネント」ノードの下の logout HREF を選択します。
2. 「要求ハンドラ」プロパティの省略符号ボタン (...) をクリックします。

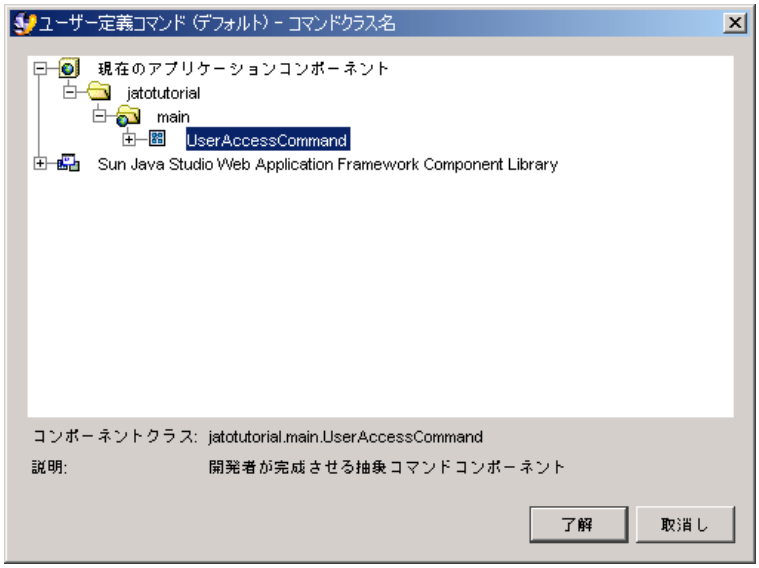
   これによって「要求ハンドラ」エディタが表示されます。



3. 「新規共有インスタンスを作成」ラジオボタンの選択リストから、「ユーザー定義コマンド (デフォルト)」を選択します。
4. 名前を userAccessCommand に変更します。
5. エディタ下部の「コンポーネントプロパティ」タブを選択します。

6. 「コマンドクラス名」プロパティの省略符号ボタン (...) をクリックします。

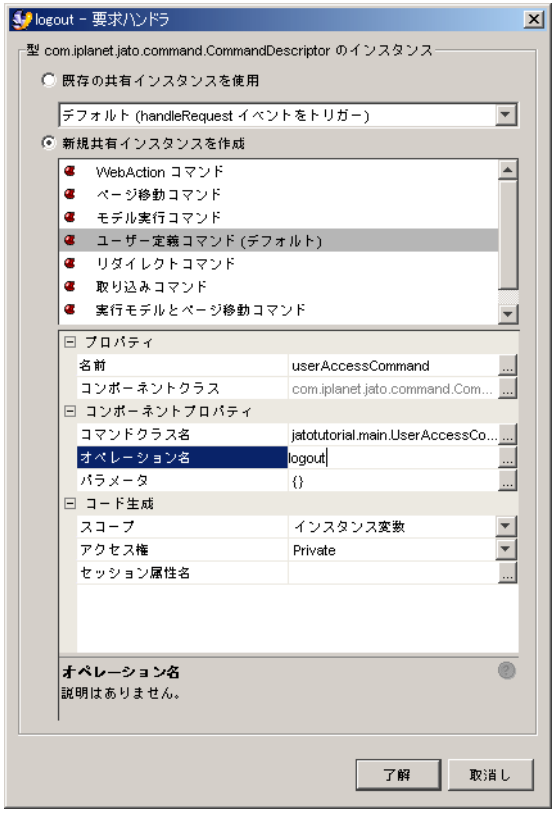
   「コマンドクラス名」選択のダイアログが表示されます。



7. 「現在のアプリケーションコンポーネント」ノード、「jatoturial」、「main」の順に展開します。
8. 「UserAccessCommand」コマンドコンポーネントを選択します。
9. 「了解」をクリックします。
10. オペレーション名を「DEFAULT」から「logout」に変更します。

    コードで UserAccessCommand クラスの execute メソッドに実装したものを思い出してください。このコードには、オペレーション名として「login」または「logout」のいずれかを期待する if/else ブロックがあります。

    これらは大文字と小文字を区別するため、正しく設定する必要があります。正しく設定しないと、このコマンドのテスト実行時に CommandException (未知のオペレーション名) を受信します。



11. 「了解」をクリックして、logout HREF の「要求ハンドラ」プロパティの設定を完了します。

    有効な顧客番号を使ってログインすると、顧客ページが表示されます。そして、ログアウトリンクが表示されます。これをクリックすると、logout オペレーション名が UserAccessCommand に渡されて、ユーザーのセッションが無効になり、ログインページにログアウトメッセージが表示されます。

顧客 JSP の HREF タグの書式設定

「logout」HREF フィールドを CustomerPage に追加したときに、HREF タグが `CustomerPage.jsp` ファイルに追加されています。ただし、リンクは、HREF のデフォルト名である「href1」を使って表示されます。これは、要求されたテキストではありません。

1. 「CustomerPage」の下の「JSP ページ」ノードを展開します。

2. 「CustomerPage」JSP ノードをダブルクリックして、エディタウィンドウで JSP を開きます。

3. logout HREF タグを探して、「href1」の代わりに「Logout」と表示するように本体の内容部分を変更します。

```
<jato:href name="logout">Logout</jato:href>
```

「useViewBean」タグ間で入れ子になり、HTML の本体セクションの一部 (「body」タグ間) である限り、HREF タグは任意の場所に配置できます。

ボタンと異なり、HREF はフォームの一部である必要はないため、「form」タグ (<jato:form>) の外側に配置できます。

第 15 章

3.3: ログイン/ログアウトコマンドコンポーネントのテスト実行

この章では、Web アプリケーションフレームワークアプリケーションの実行方法を説明します。

作業 3: ログイン/ログアウトコマンドのテスト実行

重要：PointBase ネットワークサーバーが動作していることを確認してください。動作していない場合は、以下の手順で PointBase ネットワークサバーを IDE で起動できます。

1. **メニューオプションの「ツール」>「PointBase ネットワークサーバー」>「サーバーを起動」を選択します。**

   新しいクラスを作成し、他の 2 つのクラスに変更を加えているため、アプリケーションをコンパイル、配備することが必要です。

2. **「アプリケーションクラス」ノードを右クリックして、「すべてコンパイル」アクションを選択します。**

3. **Sun Java System Application Server 上で実行している場合は、変更を加えたときにアプリケーションを配備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アプリケーション」ノード (Jato Tutorial) を選択し、Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。

4. **「LoginPage」ノードを選択し、「ページを実行 (再配備)」ボタンをクリックします。**

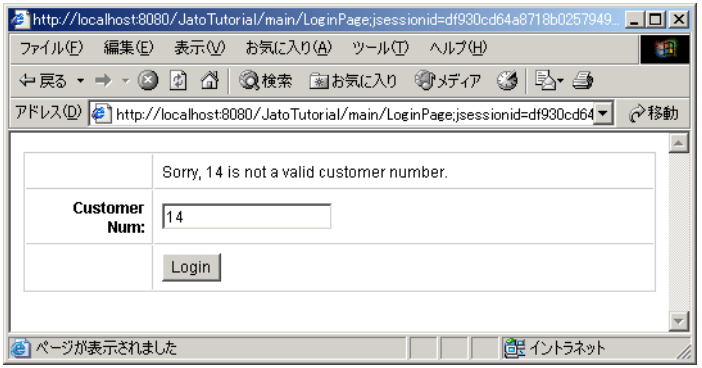
   この実行および再配備オプションを使用すると、サーバーがすべての変更を拾い出してキャッシュされたリソースを使用しないように、サーバーが再起動します。

   デフォルトのブラウザでアプリケーションが起動します。

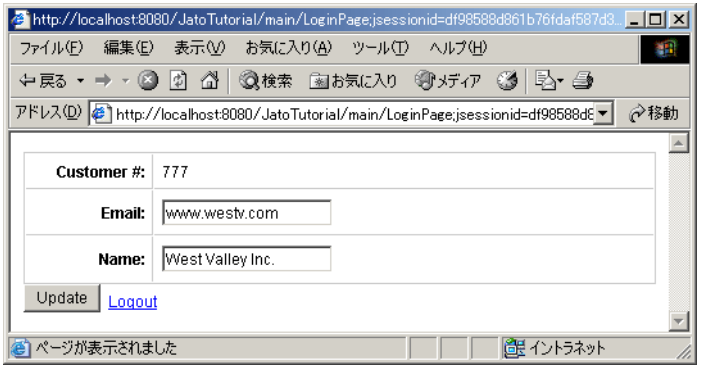
注 – 3.1 と 3.2 で、ログイン検査に 3 つの顧客番号をハードコードしました。新しい UserAccessCommand は、データベースに対して、入力された顧客番号の妥当性検査を行います。

サンプルの PointBase データベースの有効な顧客番号は、1、2、3、25、36、106、149、409、410、722、753、777、863 です。

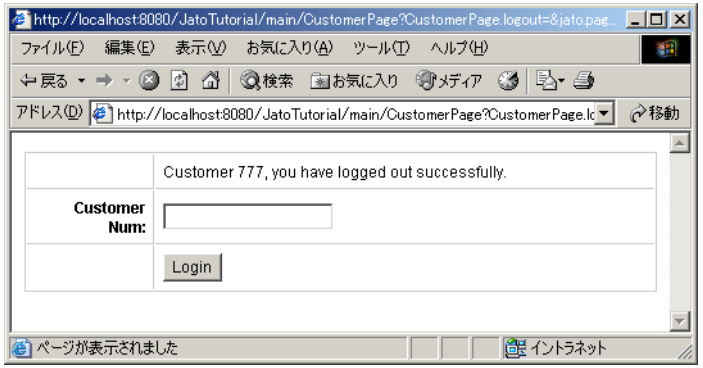
5. 最初に無効な顧客番号を入力します。



6. 有効な顧客番号を入力します。



7. 「ログアウト」のリンクを試してみます。



第 16 章

4.1: Web サービスモデルの作成準備

この章では、Web サービスを介してデータにアクセスするようにアプリケーションを拡張する方法を説明します。Web サービスモデルウィザードを装備した IDE バージョンを実行していることが必要です。また、Web サービスへの通信に干渉するプロキシやファイアウォールのないインターネット接続を装備していることが必要とされます。

Web サービスベースのモデルと、そのモデルのデータを表示するページを追加することで、既存アプリケーションを拡張できます。まず、Web サービスのモデルを構築するためのリソースをダウンロードして、この Web サービスのユーザーとして登録する必要があります。

作業 1: Web サービスのユーザー登録とダウンロード

Web サービス SDK のダウンロード

Web サービスモデルを作成するために Web アプリケーションフレームワークが必要とする WSDL ファイルを含む、Google Web サービスソフトウェア SDK をダウンロードします。

1. 以下の URL で Google Web サービス SDK をダウンロードします。
   `http://www.google.com/apis/download.html`
2. 合意事項に同意します。
3. 「ダウンロード」ボタンをクリックします。

4. **ハードディスクドライブにファイルを保存します。**

   zip ファイルを開いて、`googleapi/GoogleSearch.wsdl` ファイルをアプリケーションの lib ディレクトリ (`.../JatoTutorial/WEB-INF/lib`) に抽出します。zip ファイルには、3 つのバージョンのこのファイルが含まれています。「dotnet」以外のディレクトリにあるファイルを 1 つだけ取得してください。Web サービスモデルの構築に必要なのはこれだけです。

注 – アプリケーションファイル構造に新しいファイルをコピーすると、IDE の状態の更新にかなり長い時間がかかる場合があります。

IDE が新しいファイルを認識するのに時間がかかり過ぎると思われる場合は、「ファイルシステム」または「プロジェクト」タブで lib ディレクトリを右クリックし、「フォルダを再表示」アクションを選択してください。

Web サービスの使用登録

Google Web サービスを使用するには、ユーザーとして登録して、照会ごとに Web サービスに渡されるキーを受け取る必要があります。

1. **以下の URL で Google に登録します。**

   `https://www.google.com/accounts/NewAccount?continue=http://api.google.com/createkey&followup=http://api.google.com/createkey`

2. **電子メールアドレスとパスワードを入力して、新しいアカウントを登録します。**

   アカウントを確認するための電子メールが送られてきます。アカウントの確認後、キーを示す別の電子メールが送られてきます (キーは、文字と数字からなる長い文字列です)。この電子メールは Web サービスモデルを作成するときに必要となるため、すぐに参照できるところに保管しておいてください。

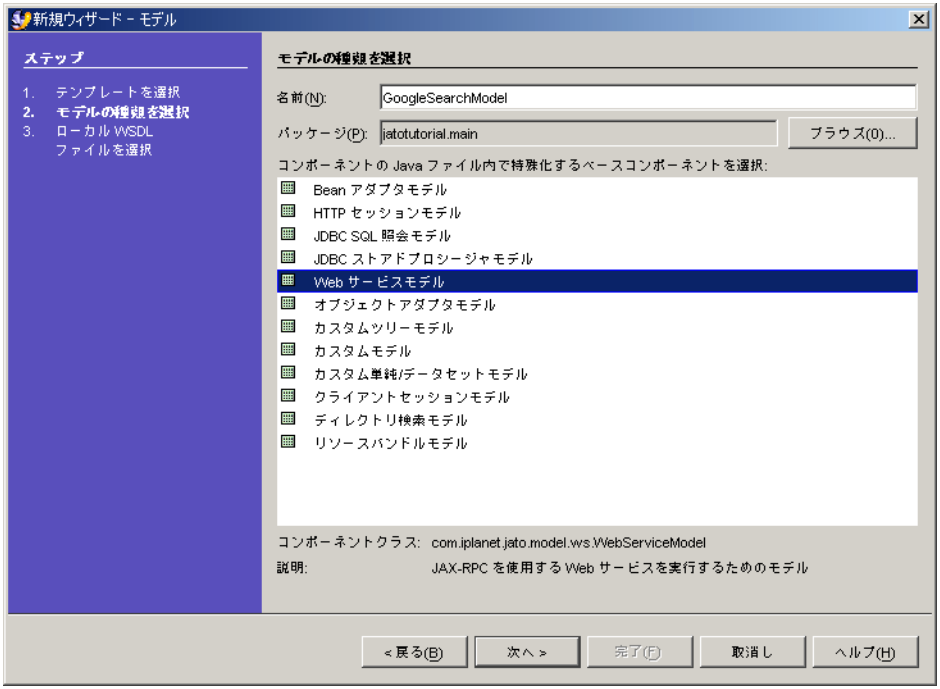
Web サービスモデルの作成

WSDL ファイルを使い、Google Web サービス経由で Google インターネット検索エンジンを使ったインターネット検索を行う Web サービスモデルを作成します。

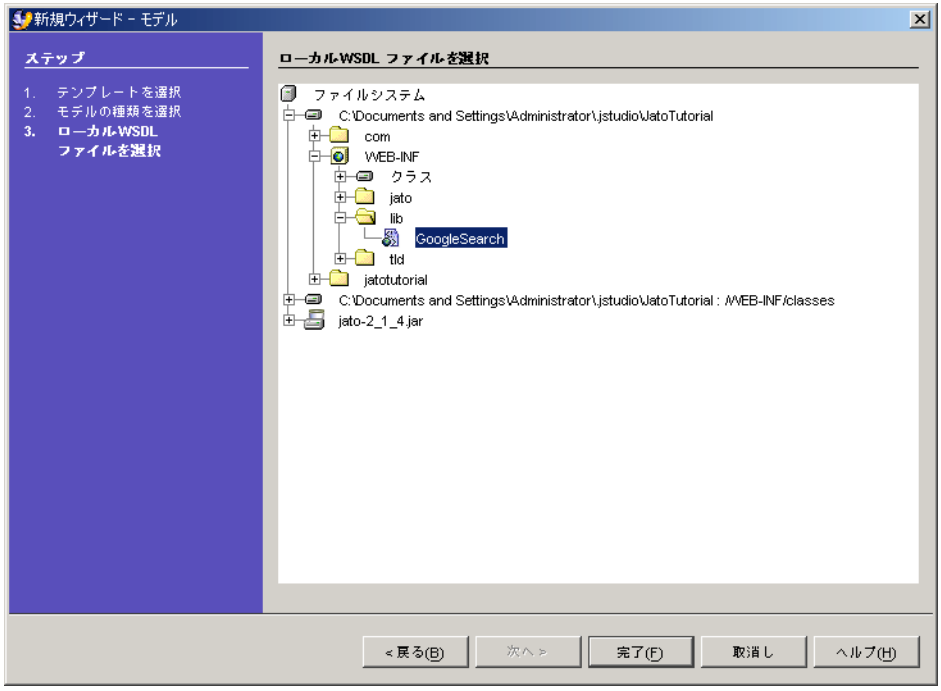
1. **「main」モジュールフォルダを選択します。**

2. **Web アプリケーションフレームワークツールバーの「モデルを追加」ボタンをクリックします。**

   「モデルの種類を選択」パネルが表示されます。



3. モデルの「名前」テキストボックスに、「GoogleSearchModel」と入力します。
4. モデルコンポーネントリストから「Web サービスモデル」を選択します。
5. 「次へ」をクリックします。
   「ローカル WSDL ファイルを選択」パネルが表示されます。

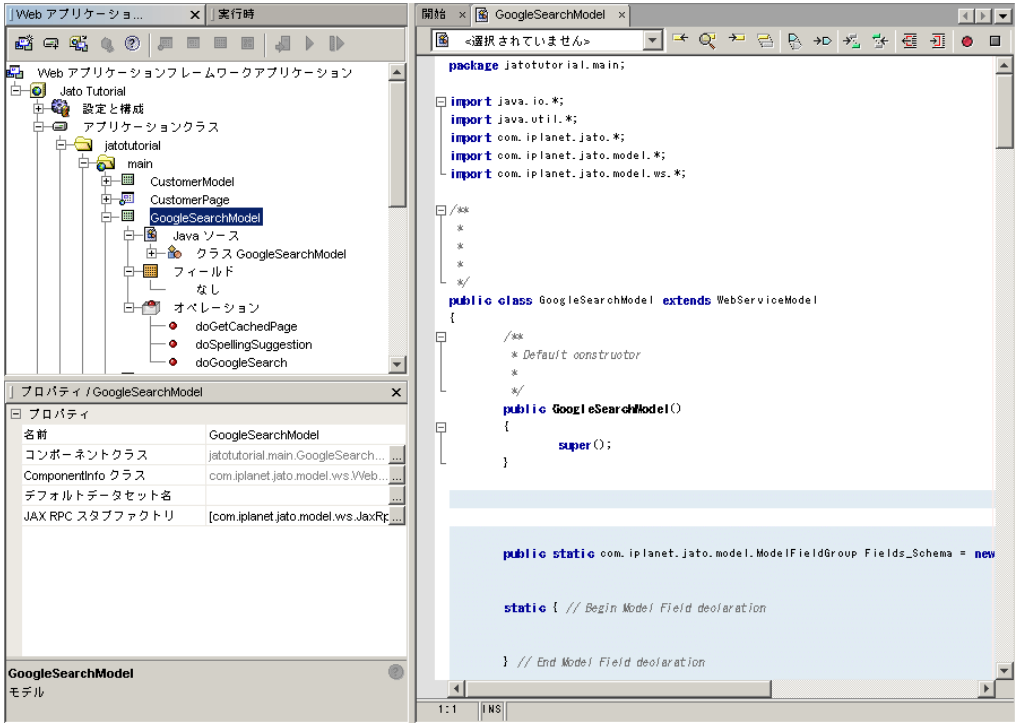


JatoTutorial アプリケーションディレクトリ構造で `lib` ディレクトリ (`JatoTutorial/WEB-INF/lib`) に移動し、GoogleSearch (WSDL ファイル) を選択します。

ファイルは、`lib` ディレクトリのサブディレクトリに格納されている場合もあります。zip ファイルから抽出したとき、これは `googleapi` というベースディレクトリに格納されました。

6. 「完了」をクリックして、Web サービスモデルを作成します。

GoogleSearchModel オブジェクトが main モジュールに作成されます。



7. 「GoogleSearchModel」ノードをダブルクリックして、GoogleSearchModel クラスのコードを表示します。

この Web サービスは、使用可能な操作をいくつか提供します。以下の作業では「doGoogleSearch」操作にのみ焦点をあてて説明します。

注 – アプリケーションファイル構造を調べると、stubs という新しいフォルダがあることが分かります。このフォルダは、Web サービスの使用をサポートするために必要なスタブクラスを格納するパッケージとして、Web サービスモデルウィザードにより作成されたものです。
これは、Web サービスモデルウィザードが提供する数多くの利点の 1 つです。Web サービスを使用するには、多数のクラスを作成する必要があります。このパッケージフォルダをブラウズして、実際にどれだけの処理が行われたかを確認してください。

これらのファイルを再度チェックする必要はありません。単調な作業はすべてウィザードによって実行されています。必要な作業は Web サービスモデルクラスの処理だけです。これも手動のコーディングは最小限で済み、ほとんどの場合はまったく必要とされません。

第 17 章

4.2: Google 検索ページの作成

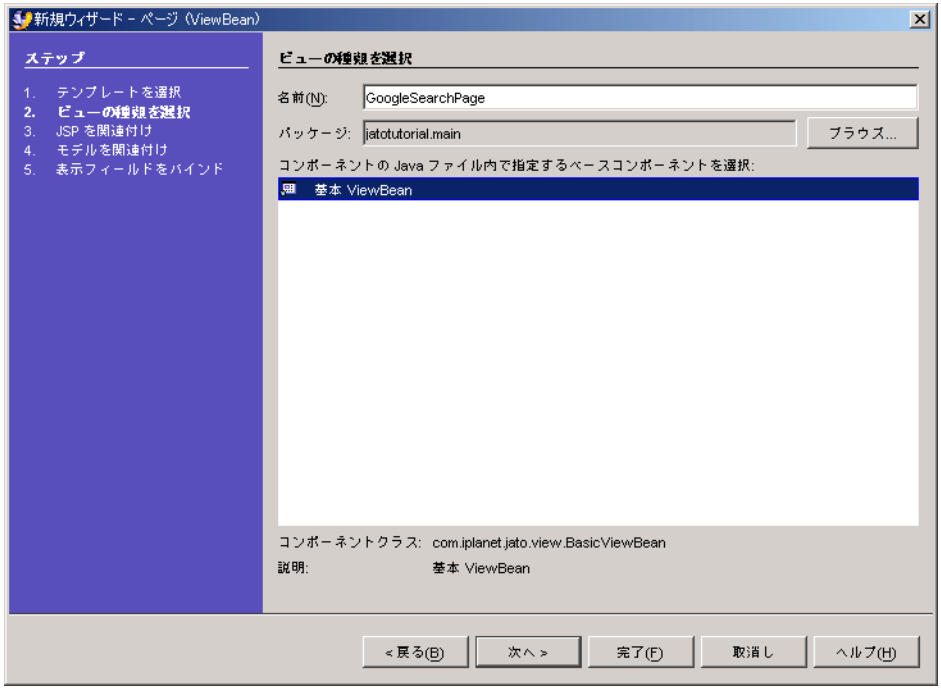
この章では、Web サービスのデータにアクセスするモデルから取得したデータを表示するページを Web アプリケーションフレームワークで作成する方法を説明します。

作業 2: Google 検索ページの作成

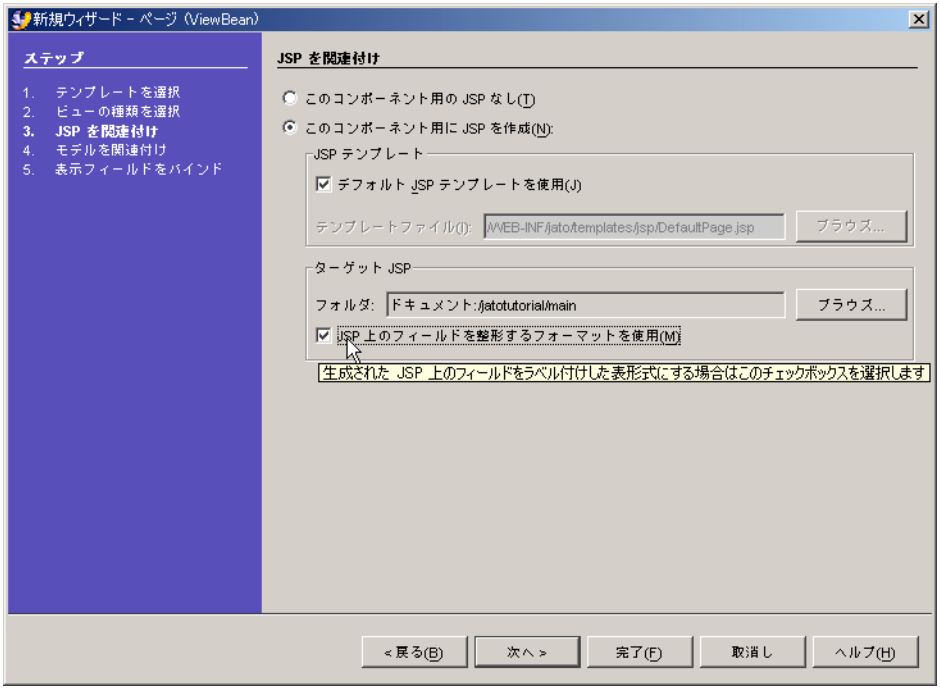
ページコンポーネントの追加

1. 「main」モジュールフォルダを選択します。
2. Web アプリケーションフレームワークツールバーの「ページを追加」ボタンをクリックします。

   「ビューの種類を選択」パネルが表示されます。



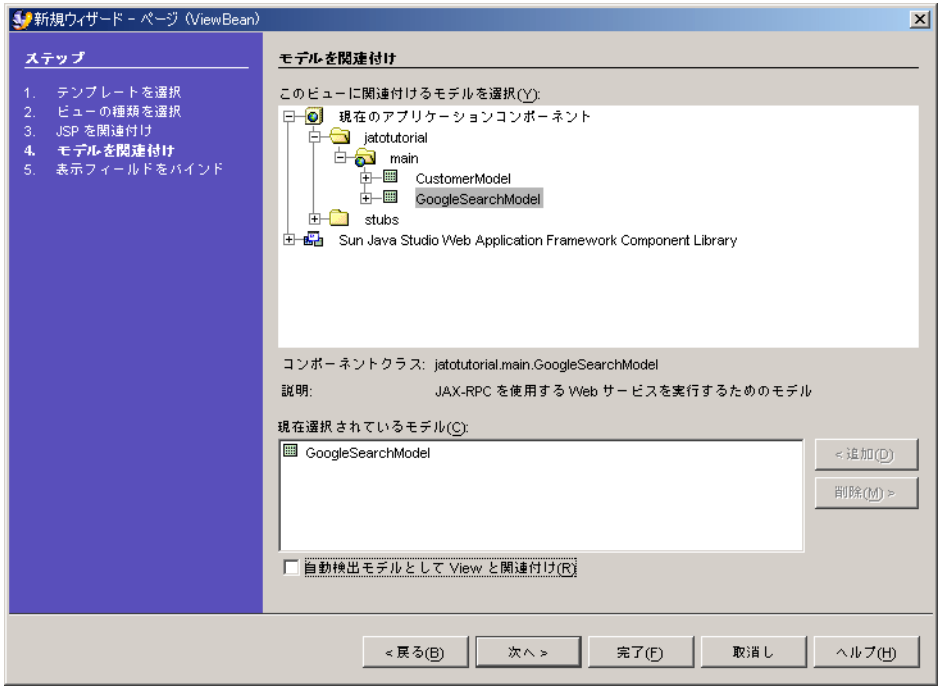
3. 「名前」フィールドに GoogleSearchPage と入力して、<デフォルト名> を置き換えます。
4. 「基本 ViewBean」を選択します。
5. 「次へ」をクリックします。
   「JSP を関連付け」パネルが表示されます。



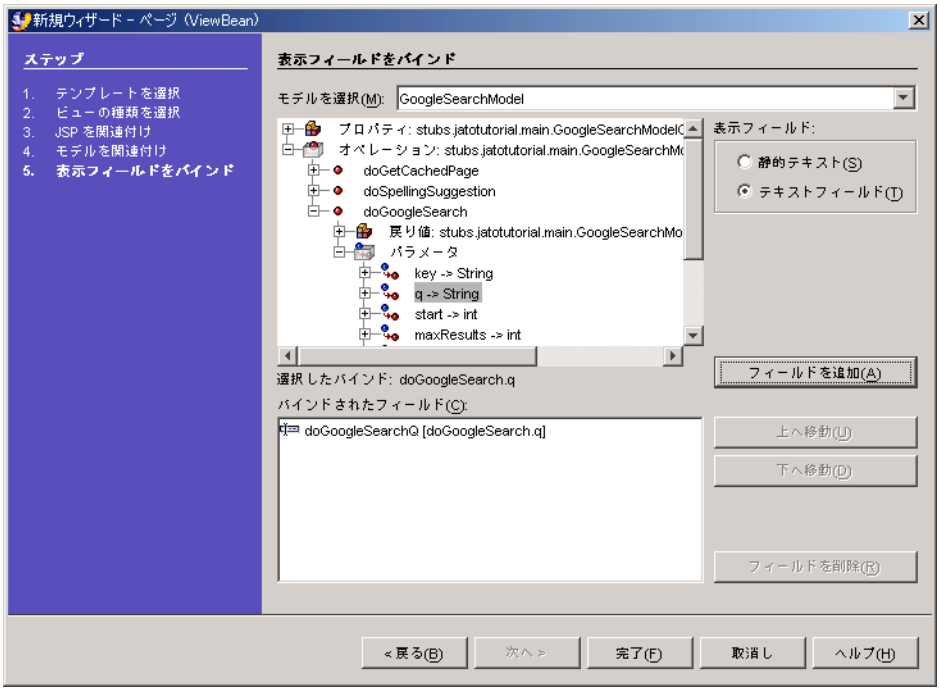
6. 「JSP 上のフィールドを整形するフォーマットを使用」オプションをチェックします。

7. 「次へ」をクリックします。

   「モデルを関連付け」パネルが表示されます。



8. 「現在のアプリケーションコンポーネント」を展開して、「jatotutorial」>「main」を表示します。
9. 「GoogleSearchModel」を選択します。
10. 「追加」をクリックします。
11. 「次へ」をクリックします。
    「表示フィールドをバインド」パネルが表示されます。



12. 最初のフィールドを追加します (上記参照)。

    a. 「doGoogleSearch」操作ノードを展開します。

    b. 「パラメータ」ノードを展開します。

    c. 「q」フィールドを選択します (照会文字列 [query string] の q: q >「String」)。

    d. 「テキストフィールド」オプションを選択します。

    e. 「フィールドを追加」をクリックします。

       q フィールドが「バインドされたフィールド」リストボックスに追加されます。

注 – このウィザードパネルでの作業は、まだ完了していません。

---

13. 以下のフィールドをテキストフィールドとして追加します (JDBC モデルフィールドとは異なり、WebService モデルフィールドは複数選択できません)。

    a. start

    b. maxResults

    c. restrict

14. 以下のフィールドを、通常のテキストフィールドではなく静的テキストフィールドとして追加します。

    (「パラメータ」ノードの上の「戻り値:...」ノードを展開します。)

    a. searchTime

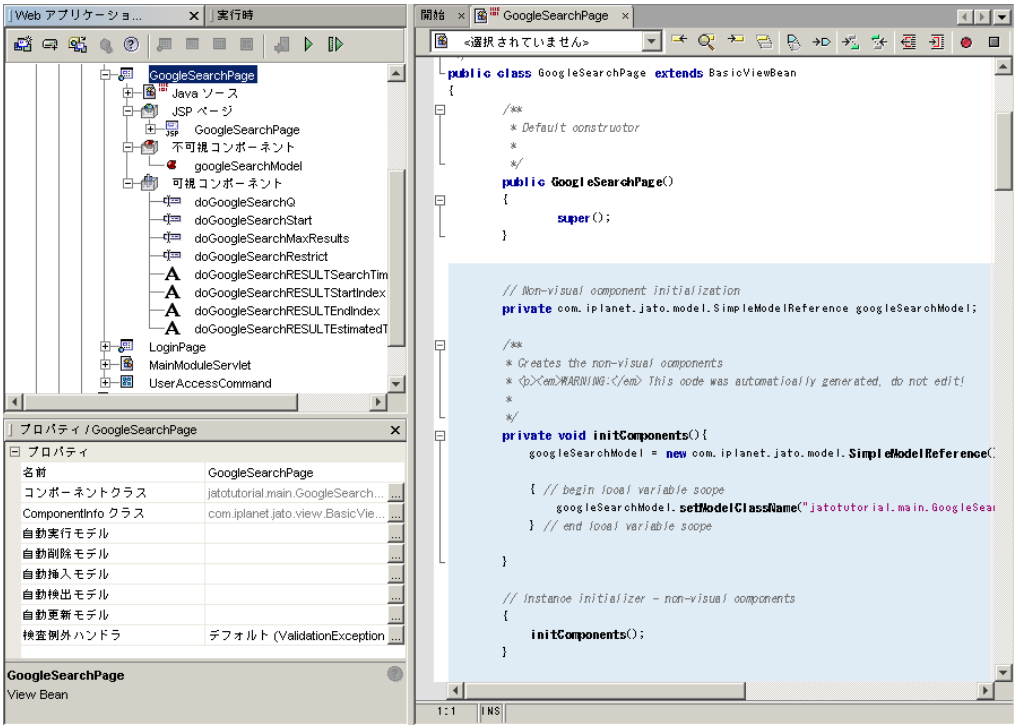
    b. startIndex

    c. endIndex

    d. estimatedTotalResultsCount

15. 「完了」をクリックします。

    これで「GoogleSearchPage」ViewBean が作成されます。



16. 短く簡単な名前にフィールド名を変更します (名前を変更するには、フィールドを選択して F2 キーを押します。

    以下の表の左側の欄に長いフィールド名を、右側の欄に短いフィールド名を示します。

doGoogleSearchQ	queryString
doGoogleSearchStart	start
doGoogleSearchMaxResults	max
doGoogleSearchRestrict	restrict
doGoogleSearchRESULTSSearchTime	searchTime
doGoogleSearchRESULTSStartIndex	startIndex
doGoogleSearchRESULTSEndIndex	endIndex
doGoogleSearchRESULTSEstimatedTotalResultsCount	estTotal

以下の表に従って、「start」および「max」テキストフィールドのプロパティを設定します。

「restrict」フィールドのプロパティは、設定する必要はありません。

重要：整数型を選択できるように、省略符号ボタン (...) をクリックして初期値エディタを起動してください。プロパティシートに値を入力すると、これは文字列として扱われます。

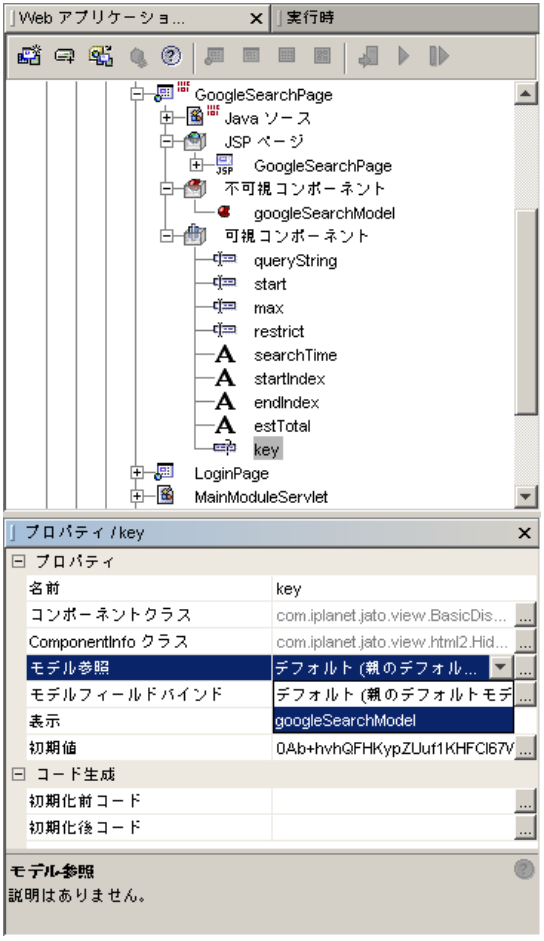
名前	初期値
start	型: Interger 値: 0
max	型: Interger 値: 5

これで、このページコンポーネントの 4 つの「検索」フィールドと 4 つの「結果」フィールドを作成しましたが、さらにいくつかの検索フィールド (Google Web サービスの必須フィールド) が必要です。これらは、一度に 1 つずつ GoogleSearchModel に追加してバインドします。これらのフィールドは、テキストまたは静的テキストフィールド以外のものとして追加します。この理由から、これらをページウィザードの外部で追加します。

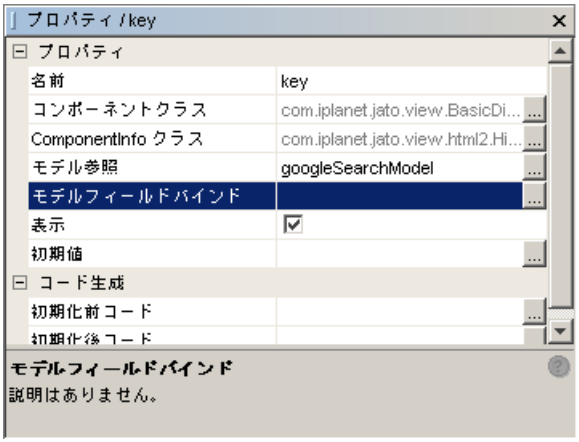
他の可視コンポーネントのページへの追加

1. 「GoogleSearchPage」を選択します。
2. コンポーネントパレットを使って、非表示フィールドを追加します。
   デフォルト名は「hidden1」です。

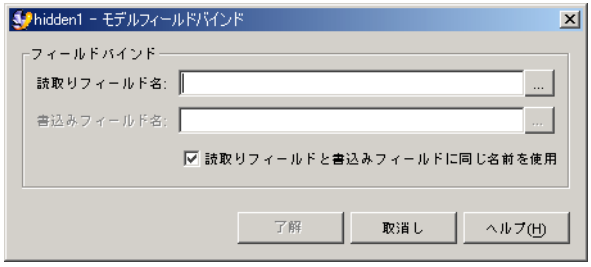
3. デフォルト名を「key」に変更します。
4. 「key」フィールドの「モデル参照」プロパティを設定します。



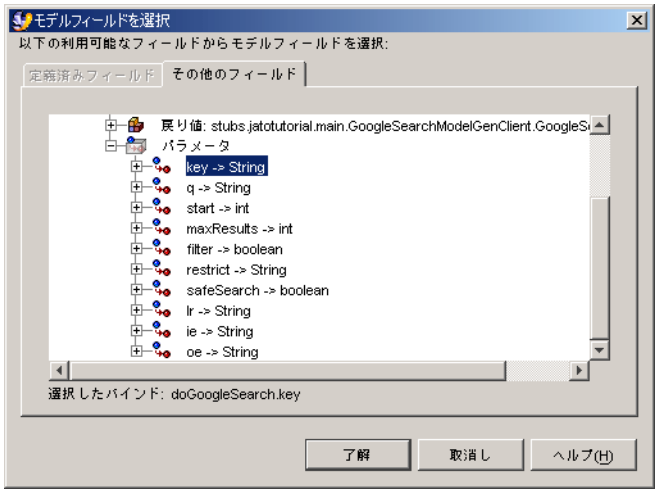
5. ドロップダウンリストから「googleSearchModel」を選択します。
6. 「モデルフィールドバインド」プロパティを設定します。



7. 省略符号ボタン (...) をクリックして、「モデルフィールドバインド」エディタを起動します。



8. このエディタで、「読取りフィールド名」プロパティの省略符号ボタン (...) をクリックします。

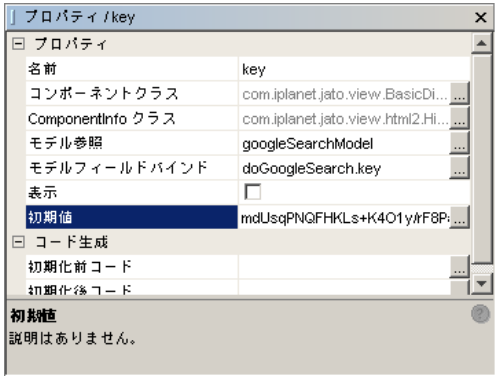


9. 「doGoogleSearch」操作ノードを展開してから、「パラメータ」ノードを展開します。
10. 「key > String」を選択します。
11. 「了解」をクリックします。

    doGoogleSearch.key モデルフィールドを使って、読取り / 書込みフィールドが生成されます。

12. 「了解」をクリックして、key 非表示フィールドの「モデルフィールドバインド」プロパティの設定を終了します。
13. Google から電子メールで送信されたキーを使って、key フィールドの「初期値」プロパティを設定します。

    「初期値」プロパティのデフォルト型は文字列です。エディタを起動する必要はありません。プロパティセルに文字列値を直接入力します。



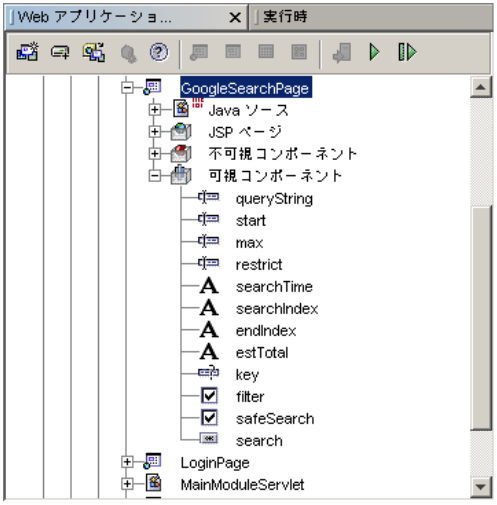
key フィールドのプロパティは上記の図のようになりますが、キーの初期値はそれぞれ異なります。

14. **コンポーネントパレットを使って、さらに 3 つの表示フィールドを追加します。**

    以下の表に、3 つの表示フィールドと必要なプロパティ設定を示します。

種類	名前	初期値	モデル参照	モデルフィールドバインド
チェックボックス	filter		googleSearchModel	doGoogleSearch/パラメータ/filter
チェックボックス	safeSearch		googleSearchModel	doGoogleSearch/パラメータ/safeSearch
ボタン	search	型: String 値: Search		

「GoogleSearchPage」ノードの構造は、以下の図のようになります。



検索ボタンの有効化

現在のところは、検索ボタンをクリックしても、何か処理を行うようには実装されていません。検索ボタンをクリックしたら、Web サービスモデルを実行し、結果を表示するためにページを再度読み込む必要があります。ここで表示されるのは、すべて以下のような統計情報です。

- 開始 / 終了索引
- 推定結果数
- 照会時間

次の作業では、実際の検索結果を一覧表示する可視コンポーネントを追加します。

ボタンの場合は、Web サービスモデルを実行してページを再度読み込むために 2 通りの方法を使用できます。数行のコードを記述するか、すべてポイントしてクリックすることによって実行します。どちらかの方法を使って実装してください。

手動コードを記述する方法

1. 「search」ボタンを右クリックします。
2. 「イベント」を選択します。
3. 「handleRequest」を選択します。

   これによって handleSearchRequest イベントスタブが GoogleSearchPage クラスに挿入されます。
4. 検索ボタンで handleRequest コードを実装します。

   次のデフォルトのコードを、この後に示すコード例に置き換えます。

   ```
   getParentViewBean().forwardTo(getRequestContext());
   ```

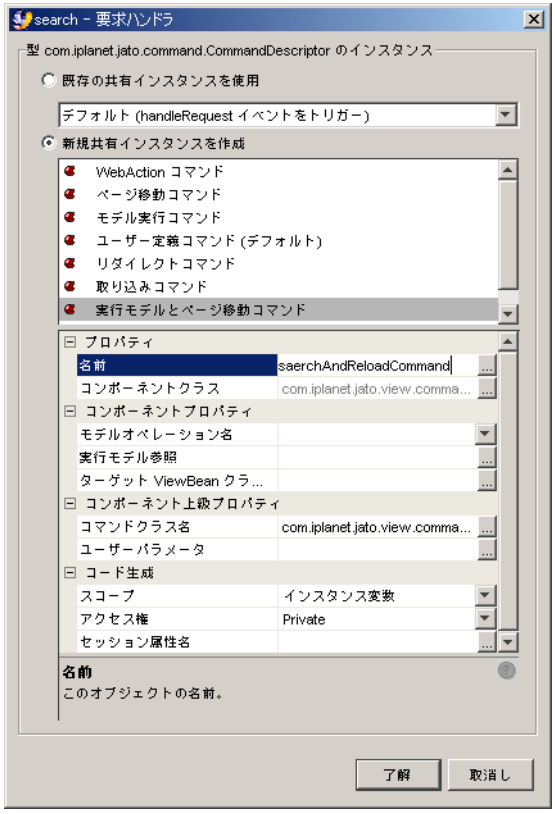
```
public void handleSearchRequest(RequestInvocationEvent event)
        throws Exception {
    // Google Web サービスモデルへの参照を取得
    GoogleSearchModel model = (GoogleSearchModel)getModel(
            GoogleSearchModel.class);

    // doGoogleSearch 操作 （モデル実行コンテキスト） を使ってモデルを実行
    model.execute(new ModelExecutionContextBase("doGoogleSearch"));

    // 照会の統計結果を示すページを再表示
    forwardTo();
}
```

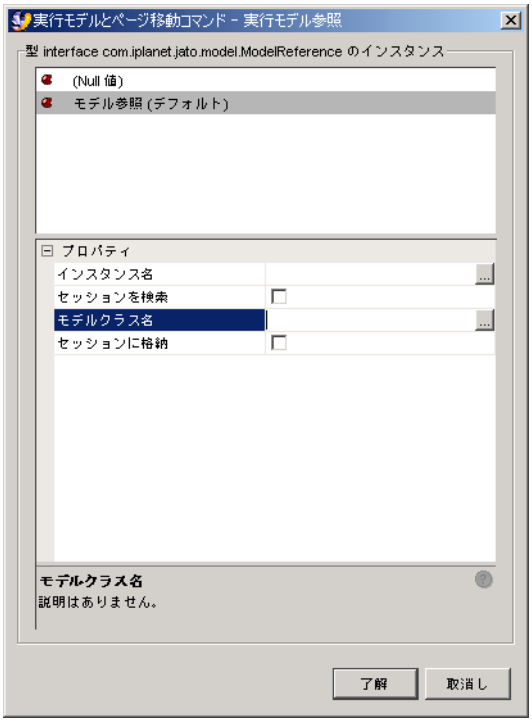
ポイントしてクリックする方法 (コードなし)

1. 「search」ボタンを選択します。
2. 省略符号ボタン (...) をクリックして、その「要求ハンドラ」プロパティのエディタを起動します。



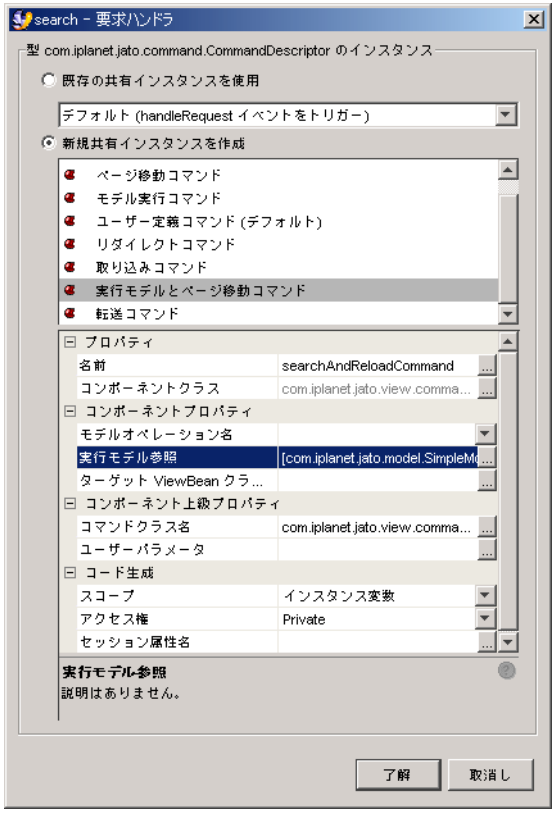
3. 「実行モデルとページ移動コマンド」オプションを選択します。
4. 「名前」を「searchAndReloadCommand」に設定します。

   「コンポーネントプロパティ」区画で、3 つの全プロパティを設定する必要があります。

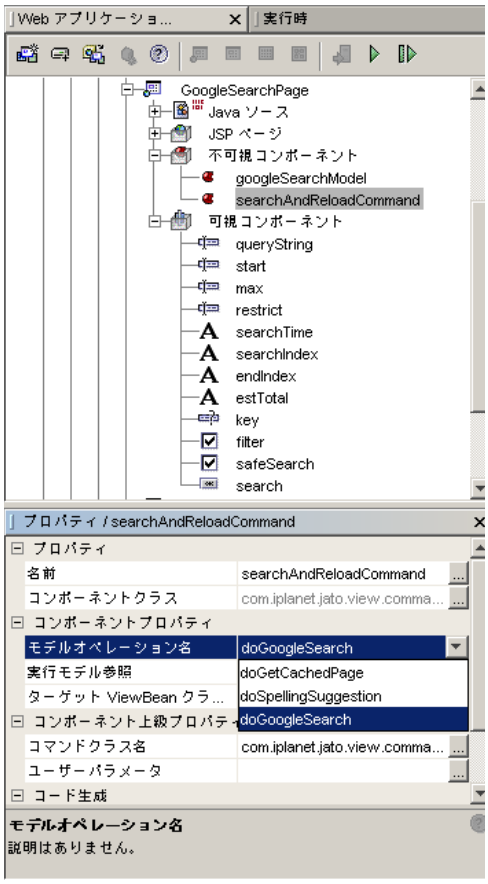


5. 「実行モデル参照」プロパティで、省略符号ボタン (...) をクリックして、そのエディタを起動します。

6. 「モデル参照 (デフォルト)」オプションを選択します。



7. 「モデルクラス名」プロパティエディタを起動します。
8. ブラウズして GoogleSearchModel を選択します。
9. 前述のエディタに戻り、「モデルクラス名」プロパティが適切に設定されたら、「了解」をクリックします。再度、「了解」をクリックして、「要求ハンドラ」プロパティエディタを終了します。
   「モデルオペレーション名」は「プロパティ」ウィンドウのドロップダウンリストから選択します。



10. 「不可視コンポーネント」ノードの「searchAndReloadCommand」を選択し、「モデルオペレーション名」のオプションリストから「doGoogleSearch」を選択します。
11. 「ターゲット ViewBean クラス名」エディタを起動します。
12. ブラウズして「GoogleSearchPage」を選択します。
13. 「了解」をクリックして、ボタンの「要求ハンドラ」プロパティの設定を完了します。

    「GoogleSearchPage」の「不可視コンポーネント」ノードに追加された「searchAndReloadCommand」が表示されます。

JSP コンテンツの書式設定

このページをテスト実行する前に、希望に応じて JSP を書式設定してください。

1. 「GoogleSearchPage」の下で、「JSP」ノードを展開し、「GoogleSearchPage JSP」をダブルクリックして、IDE のソースエディタでこれを開きます。
2. 大文字と小文字に注意して正しくフィールド名を指定します。
3. チェックボックスフィールドにラベル属性を追加して、自動的に作成されたラベルを削除します。
4. ページのタイトルを指定し、横の罫線で 2 つのセクションに分割します。上のセクションが入力フィールド、下が表示専用の静的テキストフィールドです。

   JSP/HTML コードの重要な部分を、以下に太字で示します。

```
<jato:form name="GoogleSearchPage" method="post">
<h2>Google Search</h2>
<table border=0 cellspacing=2 cellpadding=2 width="100%">
<tr>
  <td align=right valign=middle width="20%"><b>Search for:</b></td>
  <td align=left valign=middle><jato:textField name="queryString"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Start:</b></td>
  <td align=left valign=middle><jato:textField name="start"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Max Results:</b></td>
  <td align=left valign=middle><jato:textField name="max"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Restrict Search:</b></td>
<td align=left valign=middle><jato:textField name="restrict"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"><b>Filter:</b></td>
  <td align=left valign=middle><jato:checkbox name="filter" label=
"Filter?"/></td>
</tr>
<tr>
  <td align=right valign=middle width="20%"></td>
  <td align=left valign=middle><jato:checkbox name="safeSearch" label="Safe
Search?"/></td>
</tr>
</table>
<jato:button name="search"/>
<hr>
Search Time: <jato:text name="searchTime"/>
Results <jato:text name="startIndex"/>
 to <jato:text name="endIndex"/>
 of <jato:text name="estTotal"/>
<jato:hidden name="key"/>
</jato:form>
```

第 18 章

4.3: Google 検索ページのテスト実行

この章では、Web アプリケーションフレームワークアプリケーションの実行方法を説明します。

作業 3: Google 検索ページのテスト実行

いくつかのクラスに変更を加えているため、アプリケーションをコンパイルすることが必要です。

1. **「アプリケーションクラス」ノードを右クリックして、「すべてコンパイル」アクションを選択します。**

   Sun Java System Application Server 上で実行している場合は、変更を加えたときにアプリケーションを配備する必要があります。

2. **「Web アプリケーションフレームワークアプリケーション」ノード (Jato Tutorial) を選択し、Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。**

3. **「GoogleSearchPage」ノードを選択し、「ページを実行 (再配備)」ボタンをクリックします。**

   この実行および再配備オプションを使用すると、サーバーがすべての変更を拾い上げてキャッシュされたリソースを使用しないように、サーバーが再起動します。

   デフォルトのブラウザでアプリケーションが起動します。

   ページの結果部分の値は、最初はゼロになっています。

   検索によって、これらのフィールドの値が返されます。

注意 – 次のような例外が返された場合は、4.2.1 節の手順 16 (GoogleSearchPage の「自動検出モデル」プロパティから googleSearchModel を削除する) を実行していない可能性があります。

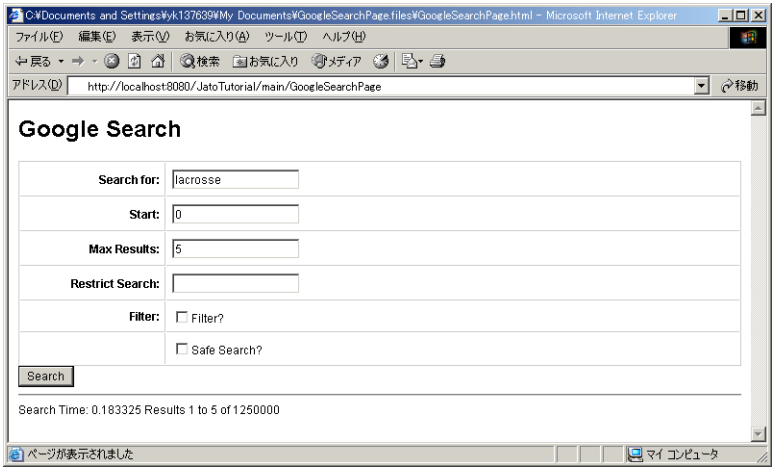
```
com.iplanet.jato.NavigationException: Exception encountered
during forward
Root cause = [com.iplanet.jato.model.ModelControlException: no
current dataset assigned yet]
```

検索の試行

1. 検索照会文字列を入力します。

   以下の図では、「Search for」に「lacrosse」が指定されています。

2. 「Search」ボタンをクリックします。



3. 他の検索を試行して、結果をチェックします。

第 19 章

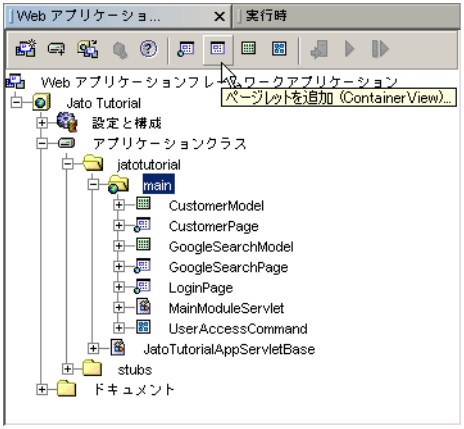
4.4: Google 検索ページへの結果リストの追加

この章では、Web サービスモデルの結果リストを表示する TiledView ページレットコンポーネントを作成する方法を説明します。TiledView は、既存のページコンポーネントに追加されます。

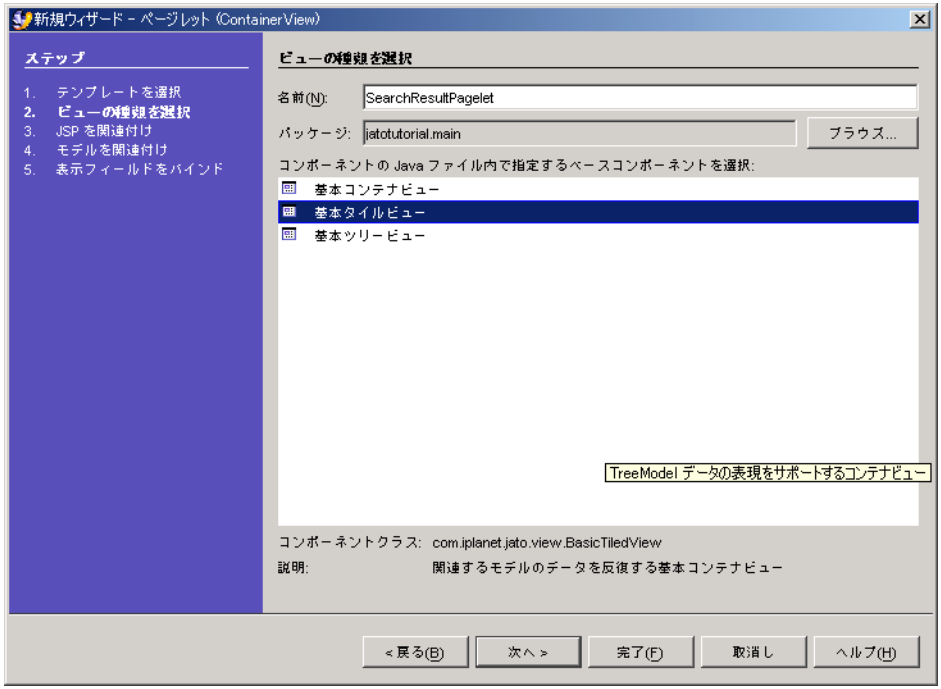
作業 4: TiledView ページレットの作成

TiledView の追加

1. 「main」モジュールを選択して、Web アプリケーションフレームワークツールバーの「ページレットを追加 (ContainerView)...」ボタンをクリックします。

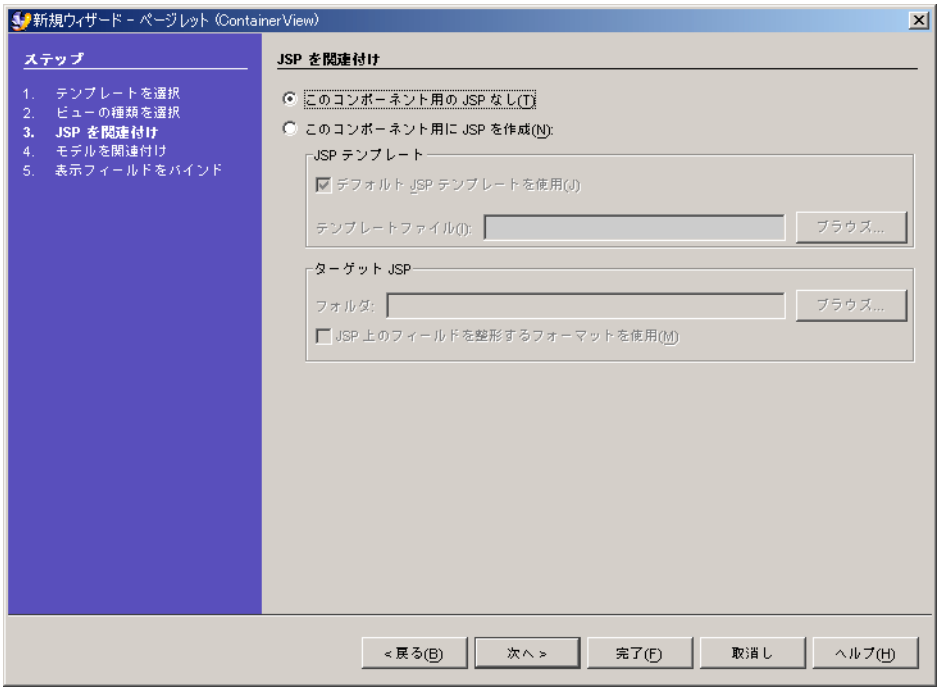


「ビューの種類を選択」パネルが表示されます。



2. 「名前」フィールドに「SearchResultsPagelet」と入力して、<デフォルト名> を置き換えます。
3. ページレットコンポーネントタイプリストから「基本タイルビュー」を選択します。
4. 「次へ」をクリックします。

   「JSP を関連付け」パネルが表示されます。

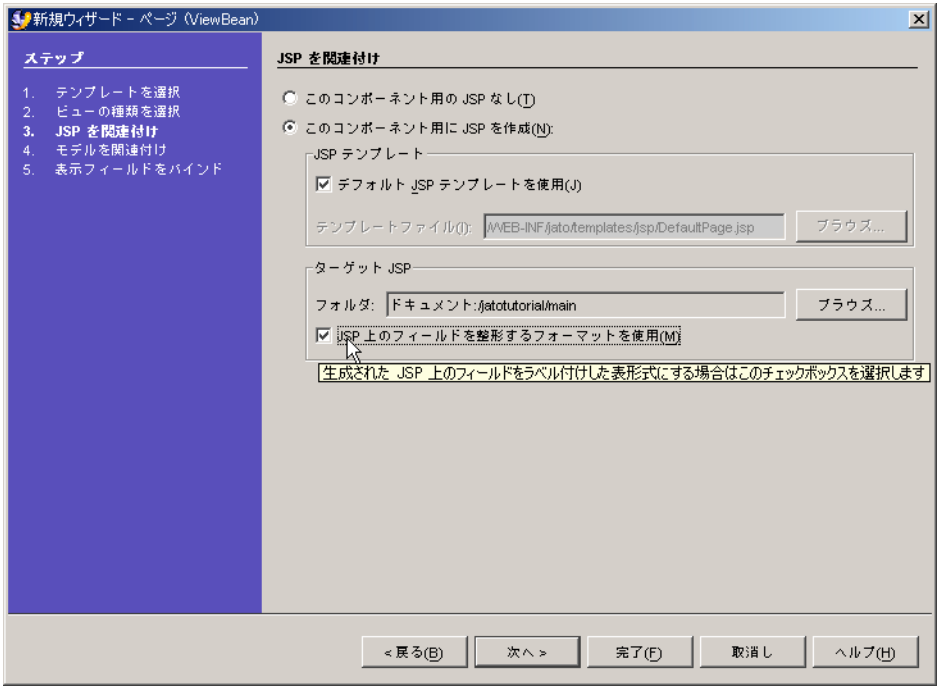


このページレットコンポーネント用の一致する JSP は、作成されません。このページレットコンポーネントの JSP タグとコンテンツは、親ページコンポーネントの JSP ページに自動的に直接追加されます。

問題は、ページレットコンポーネントの JSP を作成するかどうかということです。JSP を作成する場合もしない場合も、それぞれに長短があります。どちらに決定するかは、JSP 側でどのようにページレットコンポーネントが再利用されるかによります。どのページ (または別のページレット) がその親であるかどうかに関係なく、ページレットが同じものとして描画しようとする場合は、ページレットの JSP を作成することを推奨します。この単一の JSP ページレットファイルは、これを必要とするすべての親ページおよびページレット JSP に含まれます (JSP ファイルの include 指令)。したがって、JSP ページレットファイルに対するすべての変更は、それが含まれたすべてのものに反映されます。

ただし、親 JSP ページや親ページレットによって子のページレットの描画方法を変える場合は、JSP ページレットコンテンツをそれぞれの親 JSP ページまたはページレットファイルにインライン化してカスタマイズする必要があります。

注 – 次の図は、新規ページレットに対して JSP ファイルを作成する別の方法を示しています。ここでは、「このコンポーネント用に JSP を作成」ラジオボタンを選択し、「JSP 上のフィールドを整形するフォーマットを使用」チェックボックスを選択します。前述の方法との違いは、この方法では新規ページレットに対応した JSP フラグメントが作成される点です。対応 JSP フラグメントを持つページレットが他のページやページレットの子ビューコンポーネントとして追加されると、親コンポーネントは include 指令を使ってその JSP フラグメントに従った描画を実行します。チュートリアルのこの時点では、どちらの方法も使えます。

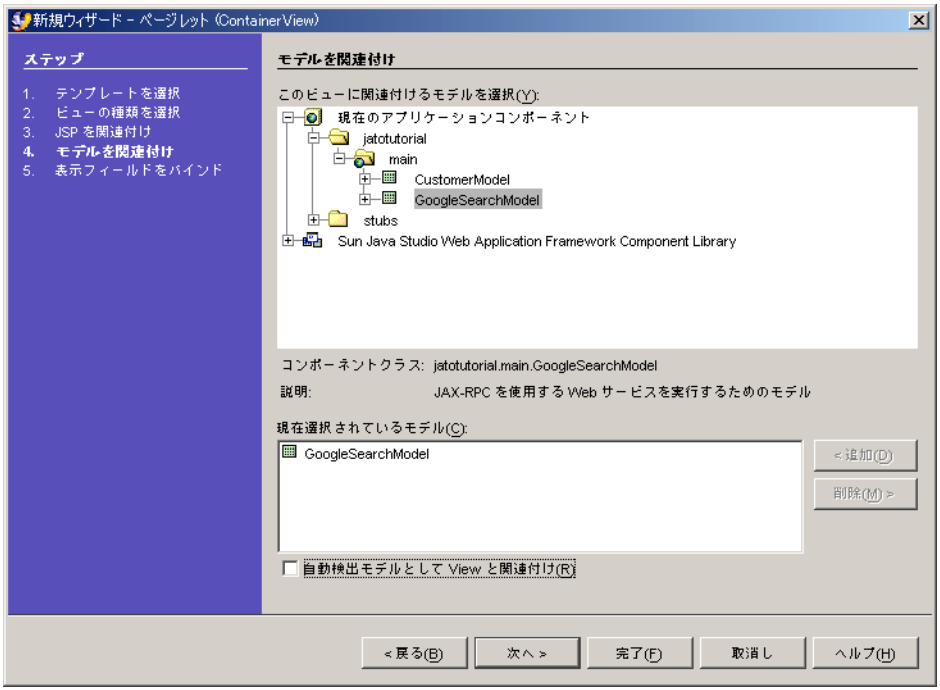


こうした方法のニュアンスはすぐには理解できないかも知れませんが、心配する必要はありません。JSP ページおよび Web アプリケーションフレームワークに習熟してくると、再使用が可能な Web アプリケーションフレームワークページおよびページレットコンポーネントの柔軟性とパワーを完全に理解できるようになります。

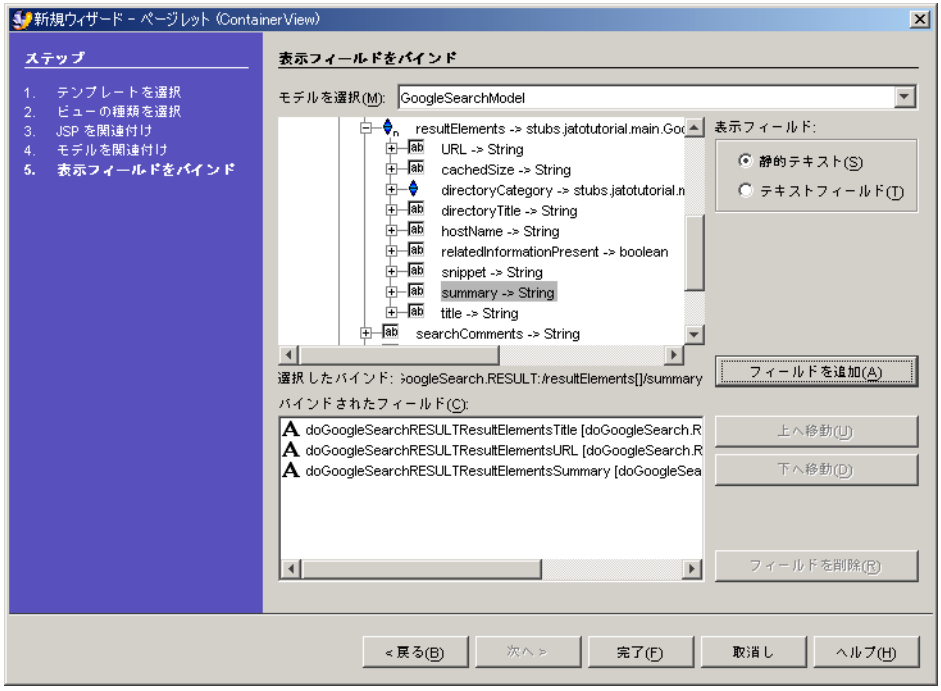
詳細は、このウィザードパネルに含まれている注を読み、『Web アプリケーションフレームワーク 開発ガイド』および『Web アプリケーションフレームワーク コンポーネント作成ガイド』を参照してください。

5. **「次へ」をクリックします。**

   「モデルを関連付け」パネルが表示されます。

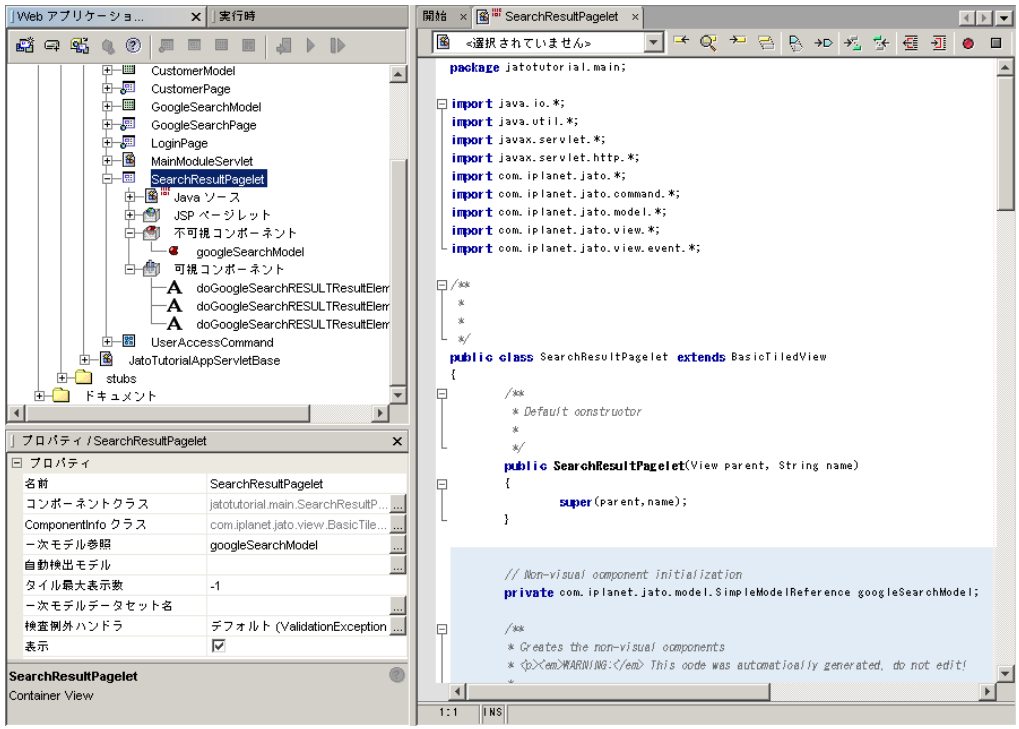


6. 「現在のアプリケーションコンポーネント」を展開して、「jatotutorial」>「main」を表示します。
7. 「GoogleSearchModel」を選択します。
8. 「追加」をクリックします。
9. 「次へ」をクリックします。
   「表示フィールドをバインド」パネルが表示されます。



10. 「doGoogleSearch」ノードを展開します。
11. 「戻り値」ノードを展開します。
12. 「resultElements」ノードを展開します。
13. 以下の 3 つの戻り値パラメータを静的テキストフィールドとして追加します。
    - title
    - URL
    - summary
14. 「完了」をクリックします。

    これで、GoogleSearchModel のリターンパラメータにバインドされた 3 つのフィールドを持つ「SearchResultsPagelet」TiledView が作成されます。



15. 短く簡単な名前にフィールド名を変更します。

    以下の表の左側の欄に長いフィールド名を、右側の欄に短いフィールド名を示します。

doGoogleSearchRESULTS**Title**	`title`
doGoogleSearchRESULTS**URL**	`url`
doGoogleSearchRESULTS**Summary**	`summary`

TiledView ページレットコンポーネントの構成

TiledView ページレットコンポーネントの 3 つのプロパティを設定する必要があります。

1. 「SearchResultsPagelet」TiledView を選択します。
2. プロパティシートで、ドロップダウンリストから「googleSearchModel」を選択して、「一次モデル参照」を設定します。

   プライマリモデルとは、TiledView が表示されるときに、その繰り返しを制御するモデルです。
3. 「タイル最大表示数」を 5 に設定します。

   これによって表示される結果の数が 5 項目に制限されます。

   -1 という値 (デフォルト) を指定すると、可能なすべての結果が検出されて表示されます。
4. 「一次モデルデータセット名」プロパティを「doGoogleSearch.RESULT:/resultElements」に設定します。

適切な一次モデルデータセット名の取得

ビューにタイル領域が装備されるように、TiledView は「DatasetModel」タイプの一次モデルを必要とします。一次モデルが「MultiDatasetModel」の場合は、TiledView が表示サイクルと送付サイクルの両方で MultiDatasetModel に自動的に「CurrentDatasetName」を設定するように、オプションで「一次モデルデータセット名」を指定できます。

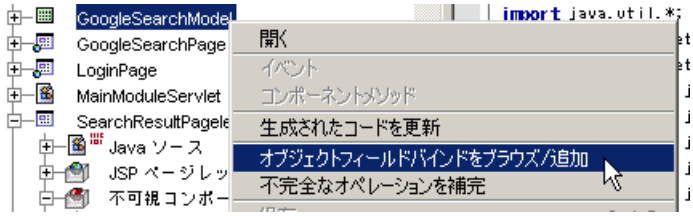
この概念は、まだ完全に理解する必要はありません。簡単に言えば、Web サービスモデルは、複数の結果セットを持つことができます。「一次モデルデータセット名」プロパティでは、特定 TiledView についてデフォルトでどれを使用するかを指定できます。

このチュートリアルでは「一次モデルデータセット名」の値が提供されていますが、自ら開発する場合はこの値をどうやって決めたらいいのでしょうか。Web サービスを十分理解している場合は、問題なく名前を決定できるでしょう。最も注意すべき事態は、タイプミスから厄介な実行時例外が発生することです。

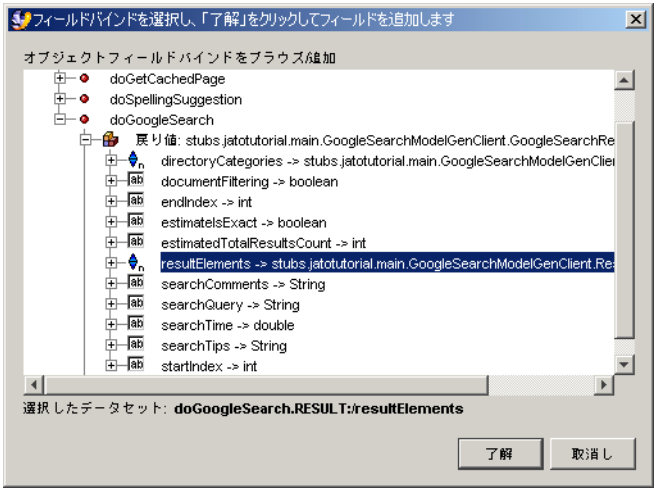
Application Framework ツールは現在、Web サービスをブラウズし「キーパス」を選択することでこの値を直接設定する方法を提供していません。しかし、この値をコピーしてからプロパティにペーストできるように、この値にアクセスする 1 回限りの Web サービスブラウズ方法があります。

まず、アプリケーションで「GoogleSearchModel」を選択してください。

1. モデルを右クリックして、「オブジェクトフィールドバインドをブラウズ/追加」を選択します。



Web サービスフィールドバインドエディタが起動します。

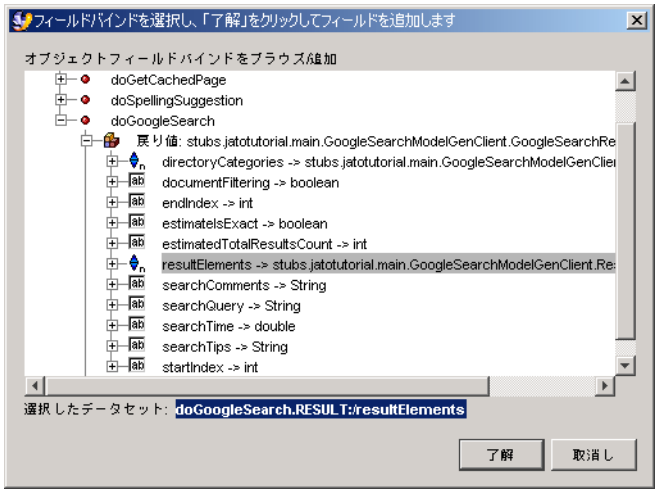


2. TiledView のフィールドがバインドされている操作パスに移動します。

3. フィールドの親ノード「resultElements」を選択します。

   このエディタ下部に、以下の「選択したデータセット」値が太字で表示されることに注目してください。「doGoogleSearch.RESULT:/resultElements」

   これは選択可能な項目としては表示されていませんが、マウスを使ってクリックしてドラッグすると選択できます。



4. これを強調表示し、Ctrl + C キーを押して、値をバッファにコピーします。
5. 「取消し」をクリックして、このエディタを終了します。

   「了解」をクリックすると、フィールドが Web サービスモデルに追加されるため、「了解」をクリックしないでください。フィールドが追加されても何の問題もありませんが、フィールドは不要です。
6. 「SearchResultsPagelet」TiledView ノードを選択して、その「一次モデルデータセット名」プロパティに値をペーストします。

ページへのページレットの追加

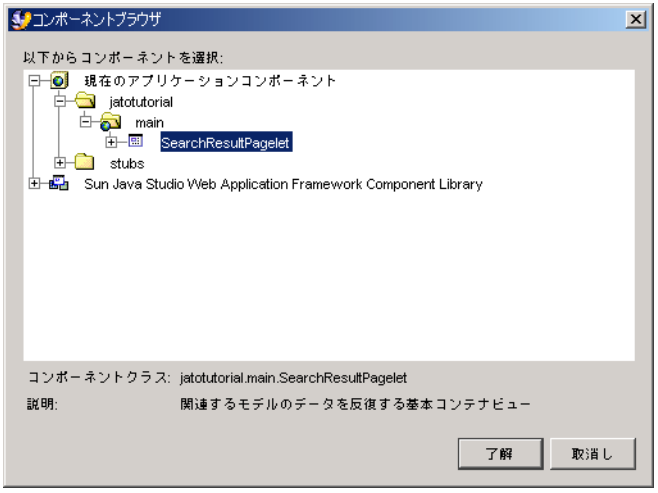
ページレットは、「ルートビュー」の助けなしでは表示できません。ページ (ViewBean) はルートビューです。ルートビューは、他のビューを含むことができるコンテナビューですが、別のコンテナビューに含めることはできません。すべてのビュー階層は、ルートビューを持つ必要があります。開発者はルートビューの下に、任意のレベル数のビューを作成できます。

これは、ドライブやディレクトリを備えた PC ファイルシステムによく似ています。ドライブ (ページに類似) は常に、すべての絶対パスの最上位になり (ルート)、パスの最上位にならないドライブはありません。

ディレクトリ (ページレットに類似) は、ドライブまたは他のディレクトリの下に含まれる必要があります。これらのディレクトリは、ドライブの下で任意の深さに入れ子にできます。

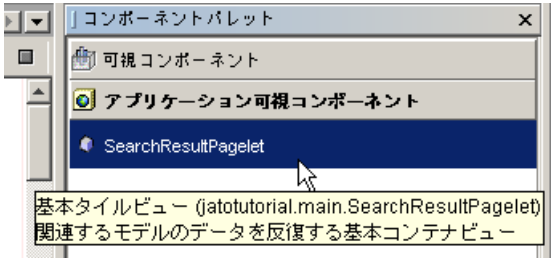
ファイル (表示フィールドに類似) は、ドライブまたはディレクトリに含まれる必要があります。ファイルには、他のファイル、ディレクトリ、またはドライブを含めることはできません。

1. 「GoogleSearchPage」ノードを展開します。
2. 「GoogleSearchPage」の「可視コンポーネント」ノードを右クリックして、「可視コンポーネント 追加」を選択します。
   これによって「コンポーネントブラウザ」ダイアログが起動します。

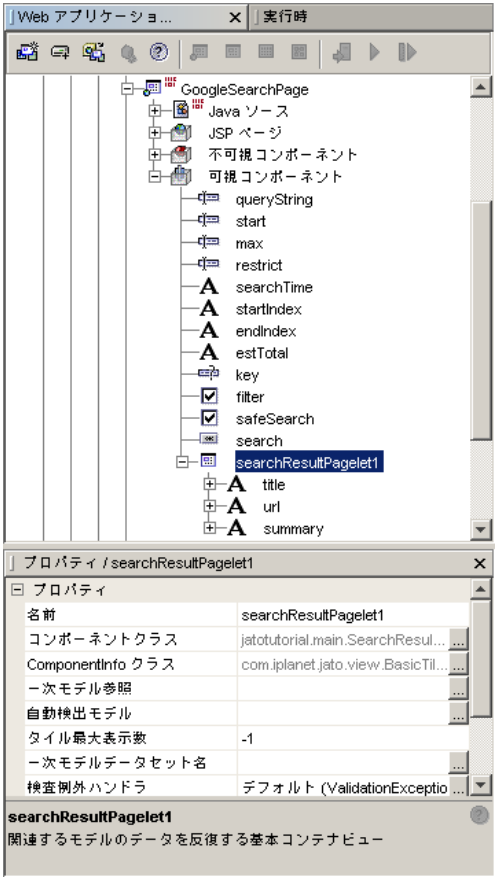


3. 「現在のアプリケーションコンポーネント」ノードを展開します。
4. 「jatotutorial」を展開します。
5. 「main」ノードを展開します。
6. 「SearchResultsPagelet」TiledView コンポーネントを選択します。
7. 「了解」をクリックします。

注 – 次の図は、コンポーネントパレットを使ってページレットを追加する別の方法を示しています。



「コンポーネントパレット」ウィンドウの「アプリケーション可視コンポーネント」セクションを選択すると、アプリケーション内でこれまで作成したすべての「ページレット」コンポーネントが表示されます。たとえば、「SearchResultPagelet」が使用可能なコンポーネントとして表示されています。手順 2-7 は、コンポーネントパレット中の「SearchResultPagelet」をマウスの左ボタンでクリックして実行することもできます。



SearchResultsPagelet TiledView は、他の可視コンポーネントと同じように、「GoogleSearchPage」の下に可視コンポーネントとして追加されます。ページレット自体は JSP を持たないことに注意してください。ページレットおよびページレットに含まれる他の可視コンポーネントに対しては、親ページコンポーネントの JSP ページにタグが追加されます。ページレットコンポーネントは複数のページコンポーネントで再使用できますが、これについてはこのチュートリアルでは説明しません。

注 – 123 ページの「TiledView の追加」では、ページレットの別の作成方法について説明しました。この項での作成方法では、新規ページレットに関連する JSP として新規 JSP フラグメントを作成しました。この方法を使って「GoogleSearchPage」を親ページとした場合は、この親ページからは追加されたページレットに対して新しいタグは追加されません。代わりに、そのページレットの JSP フラグメントに JSP include 指令が追加されます。どちらの方法でページレットを作成している場合でも、新たにページレットを追加した結果は論理的には同じになりますが、ページレットが JSP フラグメントを持っているか否かによって GoogleSearchPage JSP に加えられる変更が異なります。

JSP の書式設定

1. 「GoogleSearchPage」JSP ページノードをダブルクリックして、JSP を開きます。

   JSP の下部に、TiledView ページレットと含まれている表示フィールドタグが挿入されていますが、書式は設定されていません。

   必要なすべての HTML マークアップを追加できますが、SearchResultsPagelet TiledView に属する表示フィールドタグは `jato:tiledView` タグの内側で入れ子にする必要があります。

2. HTML の書式は、自由に設定することも、以下のコンテンツを使用することもできます。

注 – このすべてのコンテンツは、終了の JATO フォームタグ (`</jato:form>`) の直前に挿入して、現在のコンテンツはそのままにします。ここでは、body タグ間の全コンテンツが示されています。ただし、追加する必要があるのは、太字のコードのみです。

```
<body>
<jato:form name="GoogleSearch" method="post">

<jato:hidden name="key"/>
<h1>Google Search</h1>

<h2>Search Criteria</h2>
Search for: <jato:textField name="queryString"/><br>
Start: <jato:textField name="start"/><br>
Max results: <jato:textField name="max"/><br>
<jato:checkbox name="filter" label="Filter?"/><br>
<jato:checkbox name="safeSearch" label="Safe Search?"/><br>
Restrict: <jato:textField name="restrict"/><br>
<jato:button name="search"/>
<hr size="3">

<h2>Results</h2>
Search Time: <jato:text name="searchTime"/><br>
<jato:text name="startIndex"/> to <jato:text name="endIndex"/>
of <jato:text name="estTotal"/>

<jato:tiledView name="searchResultsPagelet1">
<hr size="1">
Title: <jato:text name="title" escape="false"/><br>
<a href="<jato:text name='url'/>" target="_blank">
  <jato:text name="url"/>
</a><br>
Summary: <jato:text name="summary" escape="false"/><br>
<br>
</jato:tiledView>

</jato:form>
</body>
```

上記のコードについて、以下にいくつかの点を説明します。

まず、「title」および「summary」フィールドタグの「escape」属性を見てください。デフォルトは「true」で、これは特殊文字をすべて無視します。つまり、このフィールドの値に返された HTML マークアップは、すべて一般ユーザーに表示されます。この Web サービスは、一般ユーザーが入力した照会文字列に一致するすべての文字列の周囲に、ボールドタグ (<b>) を配置します。したがって、escape=false を指定することで、このマークアップを一般ユーザーに表示するのではなく HTML マークアップとして描画するようにこのタグに指示します。あるタグを「false」に、別のタグを「true」に指定して違いをチェックすることで、この属性をテストしてください。

次に、「url」フィールドを示すタグが 2 つあることに注意してください。タグをコピー、ペーストするだけで、同じ動的データを何回も表示できます。この場合、タグの最初のインスタンスは、アンカータグの「href」属性を生成します (`<a href="...">`)。

また、「url」フィールドタグのこのインスタンスの name 属性には、単一引用符が使用されていることにも注目してください。この理由は、このタグが href 属性値の二重引用符の内側に含まれているからです。二重引用符の内側の入れ子として二重引用符を使用しても、実行時には何の問題もなく描画されますが、多くの HTML エディタでエラーメッセージが表示されます。二重引用符の内側に単一引用符を使用することで、この事態を収拾できます。

「url」フィールドのタグの 2 番目のインスタンスは、これをリンクテキストとしてユーザーに表示するために使用されます。

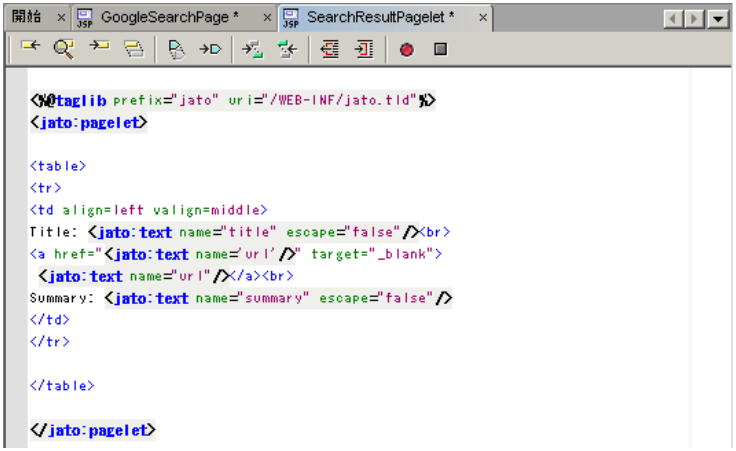
2 つの `jato:tiledView` タグ間のすべてが、1 行に 1 回の割合で、返されるデータに表示されます (この例では 5 回)。

注 – 次の 2 つの手順では、関連する JSP フラグメントを作成しないで SearchResultPagelet を作成した場合のコード例を説明します。GoogleSearchPage.jsp には、子ページレットビュー用に自動的に生成されたタグがあるため、多くの書式設定が必要になります。123 ページの「TiledView の追加」で説明した手順でページレットに関連 JSP フラグメントを作成している場合は、137 ページの「JSP とページレット JSP フラグメントの書式設定 (代替手順)」を参照してください。

JSP とページレット JSP フラグメントの書式設定 (代替手順)

1. 「GoogleSearchPage」JSP ページノードをダブルクリックし、JSP を開きます。「SearchResultPagelet」JSP ページフラグメントノードをダブルクリックし、JSP フラグメントを開きます。JSP ソースを次の図に示します。


開始
GoogleSearchPage *
SearchResultPagelet
<td align=left valign=middle><jato:textField name="restrict" /></td>
</tr>
<tr>
<td align=right valign=middle width="20%">Filter:</td>
<td align=left valign=middle><jato:checkbox name="filter" label="Filter?" /></td>
</tr>
<tr>
<td align=right valign=middle width="20%"></td>
<td align=left valign=middle><jato:checkbox name="safeSearch" label="Safe Search?"
</tr>
</table>
<jato:button name="search" />
<hr>
Search Time: <jato:text name="searchTime" />
Results <jato:text name="startIndex" />
to <jato:text name="endIndex" />
of <jato:text name="estTotal" />
<jato:hidden name="key" />
</jato:form>
<table>
<tr>
<th>Results></th>
</tr>
<jato:tiledView name="searchResultPagelet1">
<%@include file="/jatotutorial/main/SearchResultPagelet.jsp"%>
</jato:tiledView>
</body>
</html>
</jato:useViewBean>
51:1
INS


開始
GoogleSearchPage *
SearchResultPagelet *
<%@taglib prefix="jato" uri="/WEB-INF/jato.tld"%>
<jato:pagelet>
<table>
<tr>
<td align=left valign=middle>
Title: <jato:text name="title" escape="false" />
<a href="<jato:text name='url' />" target="_blank">
<jato:text name="url" />
Summary: <jato:text name="summary" escape="false" />
</td>
</tr>
</table>
</jato:pagelet>

123 ページの「TiledView の追加」での「SearchResultPagelet」の作成操作では、新規ページレットに関連した JSP フラグメントを作成するとともに自動的に書式を設定する (「JSP 上のフィールドを整形するフォーマットを使用」) チェックボックスも選択しました。前出の図を参照してください。この図では「GoogleSearchPage」が HTML の表を表すタグと表のヘッダータグとともに更新されています。また、ページレットの JSP フラグメント用の include 指令とともに「searchResultPagelet1」タイルビューも自動的に追加されています。自動的に設定された変更をこのまま使用することも、検索結果を見やすく表示するために両方の JSP ファイルを変更することもできます。

第 20 章

4.5: Google 検索ページのテスト実行

この章では、Web アプリケーションフレームワークアプリケーションの実行方法を説明します。

作業 5: 結果を使った Google 検索ページのテスト実行

新しいクラスを作成し、他の 2 つのクラスに変更を加えているため、アプリケーションをコンパイルして配備することが必要です。

1. **「アプリケーションクラス」ノードを右クリックして、「すべてコンパイル」アクションを選択します。**

   Sun Java System Application Server 上で実行している場合は、変更を加えたときにアプリケーションを配備する必要があります。

2. **「Web アプリケーションフレームワークアプリケーション」ノード (JatoTutorial) を選択し、Web アプリケーションフレームワークツールバーの「配備」ボタンをクリックします。**

3. **「GoogleSearchPage」ノードを選択し、「ページを実行 (再配備)」ボタンをクリックします。**

   この実行および再配備オプションを使用すると、サーバーがすべての変更を拾い上げてキャッシュされたリソースを使用しないように、サーバーが再起動します。

   デフォルトのブラウザでアプリケーションが起動します。

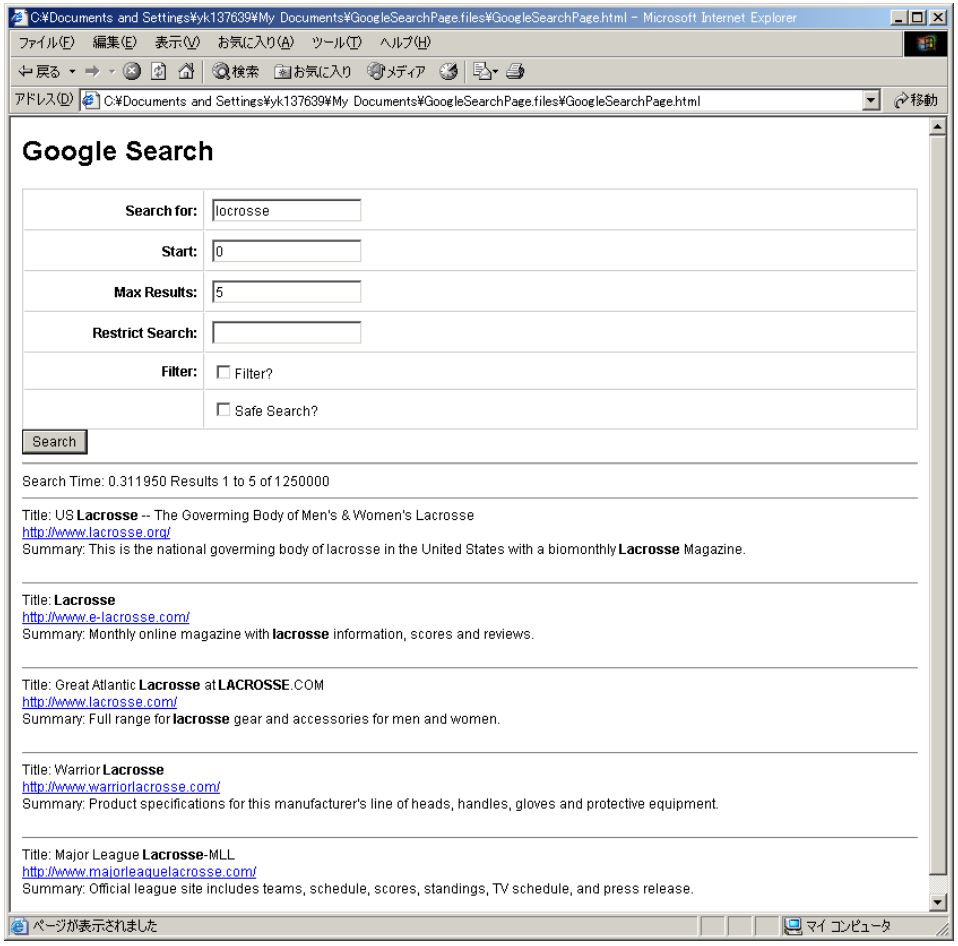
   ページの結果部分の値は、最初はゼロになっています。検索により、照会文字列の条件を満たすフィールドの値と 5 つのリンクのリストが返されます。

検索の試行

1. 検索照会文字列を入力します。

   以下の図では、「Search for」に「lacrosse」が指定されています。

2. 「検索」ボタンをクリックします。



3. 他の検索を試行して、結果をチェックします。

索引

L

M

P

R

S

T

U

V

W

あ

い

お

か

き

け

こ

さ

し

す

せ

よ

り

ろ