iSeries

WebSphere Application Server と Lotus Domino シナリオ

iSeries

WebSphere Application Server と Lotus Domino シナリオ

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： iSeries
WebSphere Application Server and Lotus Domino Scenario
Experience Report

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.8

目次

第 1 章 WebSphere(R) Application Server および Lotus(R) Domino(TM) シナリオの概要

この報告書は、iSeries(TM) システム・テスト・チームが、 WebSphere Application Server と Lotus Domino を単一のアプリケーションの一部として組み合わせた際の経験を文書化したものです。このアプリケーションでは、初期の Web プレゼンスの確立、データベースの作成、およびディレクトリー・サービスの利用のために、Lotus Domino を使用しました。 WebSphere Application Server は、動的 Web ページの作成、シングル・サインオンのセットアップ、およびセキュリティーの確保に使用しました。これらのことは、flights シナリオの一部として、複数のフェーズによって実行されました。

flights シナリオでは、市場シェアを拡大しようと努力している航空会社をシミュレートします。 flights 社は、競争力を維持するために、顧客がフライトを表示および予約できる Web サイトを持つ必要があると判断しました。同社は Web プレゼンスの確立を切望していたため、この移行を、次に示す複数のフェーズに分けて行うことを決定しました。

1. 顧客が利用可能なフライトを表示できるようにして、初期のプレゼンスを確立する。
2. 顧客がフライトをオンラインで予約できるようにして、顧客サービスを向上させる。
3. 旅行代理店との間に B2B 関係を構築し、事業を拡張する。

これらの各フェーズは、前のフェーズを土台に構築され、使用可能な最新のテクノロジーを取り入れています。各フェーズで使用した機能およびテクノロジーの詳細を、以下に示します。

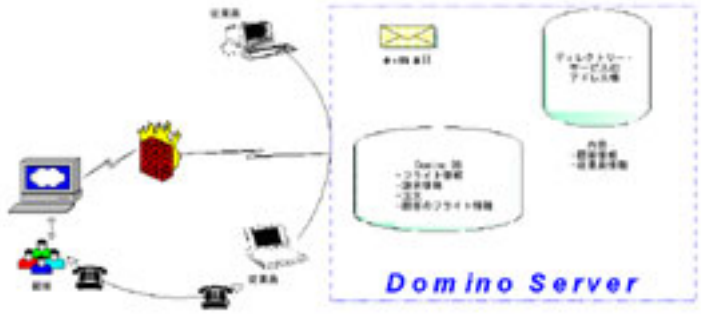
第 1 フェーズでは、Lotus Domino で提供する静的なページを使用して、Web プレゼンスを確立しました。このフェーズでは、データベース、ディレクトリー・サービス、および Lotus Domino HTTP サーバーのために Lotus Domino Server が使用されました。

第 1 フェーズでは、顧客および従業員が以下の機能を実行できるようになりました。

- 顧客
  - 利用可能なフライトの Web サイトでの表示
  - flights 従業員を呼び出してのフライトの予約
- 従業員
  - フライトの追加および削除
  - 顧客情報の追加、更新、および削除
  - フライトの予約
  - 顧客のフライト情報の検索
  - 顧客への請求書作成発行および請求情報の更新

図 1 に、第 1 フェーズを示します。

図 1 flights シナリオの第 1 フェーズ



第 2 フェーズでは、既存の Lotus Domino サーバー、データベース、およびディレクトリー・サービスを使用しました。加えて、WebSphere Application Server を使用して、顧客がフライトの予約を行える動的 Web サイトを提供しました。サーブレット、Enterprise Bean、および JSP が、シナリオのアップグレードに使用されました。 WebSphere Application Server のシングル・サインオン (SSO) およびセキュリティーが、アプリケーションのセキュリティー確保に使用されました。

第 2 フェーズでの変更により、顧客は前出の機能に加えて、以下の追加機能を実行できるようになりました。

- Web サイトを使用しての利用可能なフライトの予約
- 顧客のフライト情報の検索
- 顧客情報の更新
- オンラインでの決済

図 2 に、第 2 フェーズを示します。

図 2 flights シナリオの第 2 フェーズ

第 3 フェーズでは、旅行代理店との B2B 関係を構築します (まだ実施されていません)。この関係により、顧客は旅行代理店を通してフライトを予約できるようになります。

Lotus Domino 環境、WebSphere Application Server 環境、およびこの両者間の相互運用性についての詳細は、以下のセクションを参照してください。

5 ページの『第 2 章 Lotus(R) Domino(TM) 環境の概要』
このセクションでは、Lotus Domino 環境全体を説明します。環境の概要、環境全体を通してのワークフロー、およびこの環境の作成と使用を通して得られた主要な発見事項が記載されています。 27 ページの『第 3 章 WebSphere(R) Application Server 環境の概要』
このセクションでは、WebSphere Application Server 環境全体を説明します。環境の概説、環境全体を通してのアプリケーション・フロー、およびこの環境の作成と使用を通して得られた主要な発見事項が記載されています。

87 ページの『第 6 章 WebSphere(R) Application Server と Lotus(R) Domino(TM) との相互運用性の概要』
このセクションでは、WebSphere Application Server と Lotus Domino 環境の相互運用性について説明します。このセクションには、概要、シングル・サインオンのセットアップに関する詳細、および WebSphere Application Server と Lotus Domino の併用を通じて得られた主要な発見事項が記載されています。

第 2 章 Lotus(R) Domino(TM) 環境の概要

Lotus Domino 環境は、iSeries(TM) システム・テストの flights シナリオの作業における第 1 フェーズでセットアップされました。第 1 フェーズの目的は、flights 社がビジネスを拡張できるように Web プレゼンスを速やかに確立することと、信頼性、可用性、および拡張性に優れ、他のアプリケーションと容易に統合できるソリューションを提供することでした。

これらの要件をもとに、flights 社は eServer iSeries 上での Lotus Domino の使用を選択しました。 flights 社は Lotus Domino を使用して静的 Web ページを提供することにより、顧客および従業員に対する Web プレゼンスを迅速に確立することができました。加えて、Lotus Domino により、 flights 社はワークフロー処理を利用できるようになり、また他の数多くのプラットフォームとの相互運用性が提供されました。

flights 社の Lotus Domino アプリケーションには、顧客情報、フライト情報、旅程情報、および請求情報が含まれています。アプリケーションは、情報を作成および維持する、さまざまなフォームやビューを使用します。フライト情報は、提供されているフライトで構成されており、到着日、出発日、時刻、都市、およびフライトごとのファースト・クラスとエコノミー・クラスの座席数を含みます。顧客のデータと従業員のデータの両方を安全に管理するために、 Lotus Domino の Lightweight Directory Access Protocol (LDAP) ディレクトリーを使用します。

flights 社は、Lotus Domino のフレームセットを使用して、2 つの Web インターフェースを提供します。1 つは顧客用のインターフェース、もう 1 つは従業員用のインターフェースです。

flights の顧客は、以下の操作を行うことができます。

- フライトの表示
- flights 従業員を呼び出してのフライトの予約

flights の従業員は、以下の操作を行うことができます。

- フライトの追加、表示、および削除
- 都市コードの追加および削除
- 顧客情報の追加、更新、表示、および削除
- フライトの予約
- 顧客のフライト情報の検索
- 請求情報の表示および更新

第 2 フェーズでは、フライトの予約を含む、より多くの機能を顧客が利用できるようにするために、WebSphere(R) Application Server を使用しました。詳しくは、『WebSphere Application Server 環境の概要』を参照してください。

Lotus(R) Domino(TM) 環境のワークフロー

アプリケーション設計のポイント

Lotus Domino アプリケーションの設計時に、 flights 社は、Web インターフェースとして、フレームセットとナビゲーターのどちらを使用するかを選択する必要がありました。 flights 社では、Web サイト全体を通じて、さまざまなフォームやビューを表示可能な一貫性のある構造を提供したいと考えており、フレームには、フォーム、フォルダー、ページ、文書、およびビューを含めることができることから、フレームセットが選択されました。

また、flights 社は、データ保護の最適な方法を探してインプリメントすることも決定する必要がありました。同社では、必要なセキュリティーを提供するだけの堅固さがあり、既存のアプリケーションと容易に統合することができ、かつ、他のソフトウェア・パッケージと統合できる柔軟性を持つソリューションを求めていました。これらの要件をもとに、flights 社は Lotus Domino の Lightweight Directory Access Protocol (LDAP) を選択しました。

flights 社には既に、Java(TM) 開発のスキルを持つ従業員が数名在籍していたため、同社ではこれらの従業員のスキルを使用して、アプリケーションのバックグラウンド・エージェントを Java で作成することができました。これは、従業員のスキルを活用するという点において、有効であることが証明されました。

アプリケーションのセットアップ

Lotus Domino のインプリメント作業ではすべて、Lotus Domino Designer を開発ツールとして使用しました。 Lotus Domino Designer は、Lotus Domino アプリケーションを素早く作成および変更するために使用される、 Lotus Notes(R) クライアント・アプリケーションです。このアプリケーションは、フォーム、ビュー、エージェントなど、データベースに必要なすべてのアプリケーション構成要素を提供します。フォームは、データベース内での新規文書の作成、および現行文書の表示に使用しました。ビューを使用すると、柔軟かつ直感的な方法で文書を作成することができます。ユーザーは、文書リストの表示、さまざまな方法でのリストのソート、読み取りまたは編集のための文書のオープン、および新規文書の作成などを、簡単に行うことができます。

Lotus Domino のフォームおよびビュー

flights 社の従業員は、使用する Lotus Domino アプリケーション用に、表 1 に示されているフォームおよびビューを作成しました。

表 1 flights 社の Lotus Domino 用のフォームおよびビュー

Lotus Domino フォーム	Lotus Domino ビュー
利用可能なフライトのリスト	利用可能なフライトのリスト
スケジュールされたフライト	処理されたフライト、スケジュールされたフライト
飛行機の座席情報	飛行機の座席情報
チケット情報	アクティブなチケット、非アクティブのチケット、処理されたチケット
クレジット情報	クレジット情報

都市コード	都市コード
請求情報	請求情報
顧客番号	viewCustomerFldNumber
フライト番号	viewFlightFldNumber
請求書番号	viewInvoiceFldNumber
座席番号	viewSeatFldNumber
ステートメント番号	viewStatementFldNumber

「利用可能なフライトのリスト」フォームには、 flights 社の提供する利用可能なフライトのフライト情報が含まれます。このフォームは、新規のフライトの作成、または既存のフライトの表示に使用されます。各フライトに固有のデータは、すべてこのフォームに含まれます。同社の便が就航している都市は「都市コード」フォームに含まれ、また、飛行機の機種と空席状況は「飛行機の座席情報」フォームに含まれます。

「利用可能なフライトのリスト」ビューは、各フライトを表示し、Available Flight Number によって分類されます。

表 2 に、「利用可能なフライト」フォームのフィールドを示します。

表 2「利用可能なフライト」フォーム

名前	フィールド名	型	説明
フライト番号	AvailableFlightNumber	テキスト	フライトの番号。複数の曜日で同じ番号を使用することはできるが、同じ日に同じ番号を複数回使用することはできない。
出発時刻	DepartureTime	日付/時刻	フライトが出発都市を出発する時刻
到着時刻	ArrivalTime	日付/時刻	フライトが到着都市に到着する時刻
所要時間	Duration	数値	出発から到着までの所要時間
航空会社名	AirlineName	テキスト	フライトを提供する航空会社の名前
出発都市コード	DisplayOnlyDeparture	都市コードを基にしたダイアログ・リスト	出発都市を表す 3 文字のコード
到着都市コード	DisplayOnlyArrival	都市コードを基にしたダイアログ・リスト	到着都市を表す 3 文字のコード
飛行機種	ListAirplaneType	ダイアログ・リスト	飛行機の機種
食事	Food	ラジオ・ボタン -> (朝食 (breakfast)、昼食 (lunch)、夕食 (dinner)、軽食 (snack)、なし (none))	そのフライトで朝食、昼食、夕食、または軽食が提供されるのか、あるいは食事が提供されないのか (none) を指定する

スケジュールされている日付	ScheduledDays	チェック・ボックス -> 日曜 (Sunday)、月曜 (Monday)、火曜 (Tuesday)、水曜 (Wednesday)、木曜 (Thursday)、金曜 (Friday)、土曜 (Saturday)	スケジュールでこのフライトの運行が設定されている曜日
ファースト・クラス料金	FirstClassPrice	数値	ファースト・クラスのチケットの料金
エコノミー・クラス料金	CoachPrice	数値	エコノミー・クラスのチケットの料金
出発都市コード	DepartureCityCode	テキスト	隠しフィールド
到着都市コード	ArrivalCityCode	テキスト	隠しフィールド
表示状況	DisplayStatus	テキスト	隠しフィールド

「スケジュールされたフライト」フォームには、特定の日付の、特定のフライトに関するフライト情報が含まれます。このフォームは、特定の日付の特定のフライトの座席の予約と、座席の追跡のために使用されます。スケジュールされたフライトに固有のデータは、すべてこのフォームに含まれます。

「スケジュールされたフライト」ビューは、スケジュールされたさまざまなフライトを表示し、「日付別フライト」によって分類されます。「処理されたフライト」ビューは、到着済みのフライト、または予約できる空席の残っていないフライトをすべて表示します。

表 3 に、「スケジュールされたフライト」フォームのフィールドを示します。

表 3「スケジュールされたフライト」フォーム

名前	フィールド名	型	説明
日付別フライト	ScheduledFlightByDate	テキスト	特定の日付の特定のフライトを追跡するために使用する ID で、フライト番号にフライトの日付を結合することにより作成される
フライト番号	ScheduledFlightNumber	テキスト	フライトの番号
飛行機種	ScheduledAirplaneType	ダイアログ・リスト	飛行機の機種
ファースト・クラス空席数	FirstClassSeatsAvailable	数値	その飛行機に残っているファースト・クラスの空席の数
エコノミー・クラス空席数	CoachSeatsAvailable	数値	その飛行機に残っているエコノミー・クラスの空席の数
状況	Status	ラジオ・ボタン -> (キャンセル (Cancelled)、到着済み (arrived)、出発済み (departed)、スタンバイ (standby)、新規 (new))	フライトの状況

出発日	ScheduledDepartureDate	日付	そのフライトの出発する日付
到着日	ScheduledArrivalDate	日付	そのフライトの到着する日付 (夜間のフライトや、国際日付変更線を越えるフライトの場合は異なる可能性がある)

「飛行機の座席情報」フォームには、飛行機に固有の情報が含まれます。このフォームは、飛行機の機種と、空席数の合計をリストするために使用されます。

「飛行機の座席情報」ビューは、さまざまな飛行機を表示し、 Airplane Type によって分類されます。

表 4 に、「飛行機の座席情報」フォームのフィールドを示します。

表 4「飛行機の座席情報」フォーム

名前	フィールド名	型	説明
飛行機種	SeatAirplaneType	テキスト	飛行機の機種
ファースト・クラス座席	FirstClassSeats	数値	その飛行機に存在するファースト・クラスの座席数
エコノミー・クラス座席	CoachSeats	数値	その飛行機に存在するエコノミー・クラスの座席数
飛行機の全情報	AllAirplaneInfo	テキスト	隠しフィールド

「チケット情報」フォームには、1 つのチケットに関する情報が含まれます。請求書には、1 つ以上のチケットが含まれます。

「アクティブなチケット」ビューは、未処理のチケットを表示します。これらの注文は、処理されるとこのビューから除外され、チケットが正常に処理されたかどうかに応じて、「処理されたチケット」ビューまたは「非アクティブのチケット」ビューのいずれかに表示されます。

「非アクティブのチケット」ビューは、処理できなかったチケットを表示します。処理できなかったチケットの例としては、そのフライトのチケットが残っていないときに発注されたものや、キャンセルされたフライトのチケットなどがあります。

「処理されたチケット」ビューは、処理が完了したチケットをすべて表示します。

表 5 に、「チケット情報」フォームのフィールドを示します。

表 5「チケット情報」フォーム

名前	フィールド名	型	説明
請求書番号	InvoiceNumber	テキスト	チケット同士を結びつけるために使用する番号
フライト番号	TicketFlightNumber	テキスト	「利用可能なフライトのリスト」文書に関連付ける
チケットのクラス	TicketClass	ダイアログ・リスト	そのチケットがファースト・クラスとエコノミー・クラスのどちらであるか
座席番号	SeatNumber	テキスト	このフライトでの座席番号
チケット料金	TicketPrice	数値	この座席のチケットの料金
支払状況	PaidStatus	ラジオ・ボタン	チケットの状況
顧客番号	TicketCustomerNumber	テキスト	このチケットの料金を支払う顧客の番号
チケット状況	TicketStatus	ダイアログ・リスト	チケットの状況 (アクティブ (Active)、非アクティブ (Inactive)、または処理済み (Processed))
乗客のファーストネーム	PassengerFirstName	テキスト	割り当てられた席を利用する顧客の名前。これは、チケットを購入する人物とは異なる可能性があり、そのために顧客番号が用意されています
乗客のミドル・ネーム	PassengerMiddleName	テキスト	乗客のミドル・ネーム
乗客のラストネーム	PassengerLastName	テキスト	乗客のラストネーム
乗客の住所	PassengerStreet	テキスト	乗客の住所
乗客の市区町村	PassengerCity	テキスト	乗客の市区町村
乗客の都道府県	PassengerState	テキスト	乗客の都道府県
乗客の郵便番号	PassengerZip	テキスト	乗客の住所の郵便番号
乗客の国	PassengerCountry	テキスト	乗客の国籍
乗客の電話番号	PassengerPhone	テキスト	乗客の電話番号
日付別フライト	FlightByDate	テキスト	このチケットのフライトの日付別フライト番号

「クレジット情報」フォームには、顧客番号で示された顧客のクレジット情報が含まれます。

「クレジット情報」ビューは、顧客がフライトの料金の支払いに使用するクレジット・カード情報を表示し、「顧客番号」によって分類されます。

表 6 に、「クレジット情報」フォームのフィールドを示します。

表 6「クレジット情報」フォーム

名前	フィールド名	型	説明
カード・タイプ	CardType	ダイアログ・リスト (Mastercard、Visa、Diner's Club)	クレジット・カードの種類
クレジット・カード番号	CreditCardNumber	テキスト	クレジット・カード番号
有効期限	ExpirationDate	日付/時刻	カードの有効期限が切れる日付
顧客番号	CreditCustomerNumber	テキスト	そのカードに関連付けられている顧客番号
請求書番号	InvoiceNumber	テキスト	チケット同士を結びつけるために使用する番号

「都市コード」フォームには、サポートされる都市のリストと、各都市に対応する 3 文字の都市コードが含まれます。

「都市コード」ビューは、さまざまな都市コードを表示し、コードによって分類されます。

表 7 に、「都市コード」フォームのフィールドを示します。

表 7「都市コード」フォーム

名前	フィールド名	型	説明
市区町村	City	テキスト	都市
都道府県	State	テキスト	都道府県
国	Country	テキスト	国
コード	Code	テキスト	指定された都市を表す 3 文字のコード
都市コードの全情報	AllCityCodeInfo	テキスト	隠しフィールド
保管オプション	SaveOptions	数値	隠しフィールド - デフォルトでは文書を保管しないように 0 に設定

「請求情報」フォームには、顧客へ請求を行うための情報が含まれます。

「請求情報」ビューは、顧客に対してフライトの料金請求を行うために使用されるチケット情報を表示し、「顧客番号」で分類されます。

表 8 に、「クレジット情報」フォームのフィールドを示します。

表 8「クレジット情報」フォーム

名前	フィールド名	型	説明
請求額	BillAmount	数値	顧客が支払わなければならない金額
顧客番号	BillCustomer	テキスト	顧客番号
チケット請求書情報	BillTicketInfo	テキスト	この請求に関連したチケット

viewCustomerFldNumber、viewFlightFldNumber、viewInvoiceFldNumber、viewSeatFldNumber、および viewStatementFldNumber は、隠しビューであり、顧客番号、フライト番号、請求書番号、座席番号、およびステートメント番号の文書を表示します。新規文書を作成する際に、PostOpen イベントは文書内の現在の値を利用して、固有値を作成します。数値が作成されていると、 PostOpen イベントは値を増分して、文書に保管します。

アプリケーション・データベース内で作成されたフォームおよびビューに加えて、 flights 社は、次の表 9 に示したフォームおよびビューを Lotus Domino サーバーのアドレス帳に追加しました。

表 9 Lotus Domino フォームおよびビュー

Lotus Domino フォーム	Lotus Domino ビュー
フライト乗客	フライト顧客

「フライト乗客」フォームには、flights 社の各顧客の名前、住所情報、および顧客番号が含まれます。flights 社は、アドレス帳にある既存の「人物 (Person)」フォームのデザインを元に、自社のニーズに合わせて変更してこのフォームを作成しました。

「フライト顧客」ビューは、顧客情報を表示し、「顧客番号」で分類されます。

表 10 に、「フライト乗客」フォームのフィールドを示します。

表 10「フライト乗客」フォーム

名前	フィールド名	型	説明
ファーストネーム	FirstName	テキスト	顧客のファーストネーム
ミドル・ネームのイニシャル	MiddleInitial	テキスト	顧客のミドル・ネームのイニシャル
ラストネーム	LastName	テキスト	顧客のラストネーム
インターネット・アドレス	InternetAddress	テキスト	顧客のインターネット・アドレス
インターネット・パスワード	HTTPPassword	テキスト	flights 社の Web サイトにアクセスするための顧客のパスワード
番地	StreetAddress	テキスト	顧客の自宅の番地

市区町村	City	テキスト	顧客の市区町村
都道府県	State	テキスト	顧客の都道府県
郵便番号	Zip	テキスト	顧客の郵便番号
国	Country	テキスト	顧客の国
自宅電話番号	PhoneNumber	テキスト	顧客の自宅の電話番号
顧客番号	PersonalID	数値	顧客番号

Lotus Domino エージェント

Lotus Domino エージェントは、作業を自動化するために Lotus Domino データベースに追加される設計要素です。エージェントは、ユーザー・アクションによって開始したり、スケジュール・ベースで実行することができます。エージェントは、一般に、文書を更新および作成したり、 Lotus Domino データベースやその他のソースからデータにアクセスする際に使用されます。 Lotus Domino エージェントは、Java、LotusScript、または式言語で作成することができます。

エージェントの作成には、Lotus Domino Designer が必要です。エージェントを作成するときには、そのエージェントをいつ実行させるか、コードをどの言語で記述するか、および、どの文書の下でエージェントを実行するかを指定します。コードを記述してコンパイルすると、そのエージェントは、あらかじめ指定した所定の時刻に実行されるよう、自動的にスケジュールされます。コードを記述する際には、 Lotus Domino Designer に組み込まれたデバッグ機能を利用して、コードのデバッグに役立てることができます。

flights 社の Lotus Domino アプリケーション用に作成された Lotus Domino エージェントに関する詳細情報を、以下に示します。

- 固有都市コードのチェック
  この LotusScript エージェントは、Web ブラウザーから、「都市コード」フォームを使用して都市コードを入力として受け取ります。このエージェントは、その情報を既存の「都市コード」文書 (「都市コード」フォームによって作成されたもの) の情報と共に使用して、すべての「都市コード」文書を検索します。一致するものが見つかると、エラー・メッセージが表示されます。ユーザーは通知を受け、別の「都市コード」文書の作成が許可されます。一致するものがない場合は、その「都市コード」文書が作成されます。
- 顧客番号を無作為に作成、フライト番号を無作為に作成、および、請求書番号を無作為に作成 - Web のみ
  これらの LotusScript エージェントは、Lotus Domino の Web インターフェースを通して追加された新規の顧客、フライト、または請求書用に、固有の顧客番号、フライト番号、または請求書番号を生成します。
- 都市コードおよび利用可能なフライトのリストの削除ボタン
  この Java エージェントは、ユーザーが「都市コード」ビューまたは「フライト」ビューの「削除 (Delete)」アクション・ボタンをクリックすると実行されます。選択された「都市コード」文書または「フライト」文書を、削除済みとしてマーキングします。
- 選択済み都市コードの削除
  この Java エージェントは、スケジュール・ベースで実行されます。 CityCodeStatus が「削除 (Delete)」

となっている都市コードがあれば「都市コード」文書から削除します。また、都市コードの一致する「利用可能なフライトのリスト」、「スケジュールされたフライト」、および「チケット情報」からも文書を削除します。

- 選択済みフライトの削除
  この Java エージェントは、スケジュール・ベースで実行されます。「表示状況」が「削除 (Delete)」となっているフライトがあれば、「利用可能なフライトのリスト」文書から削除します。また、フライト番号の一致する「スケジュールされたフライト」および「チケット情報」からも文書を削除します。
- チケットの削除 - 空席情報の更新
  この Java エージェントは、ユーザーがアクション・ボタンをクリックすると実行されます。「スケジュールされたフライト」文書から選択された「日付別フライト」および「チケット・クラス」に基づき、空席の数を増分します。
- 請求情報生成
  この Java エージェントは、ユーザーがアクション・ボタンをクリックすると実行されます。選択された顧客番号に基づき、顧客の請求情報を生成します。
- 料金および座席番号生成
  この Java エージェントは、ユーザーがアクション・ボタンをクリックすると実行されます。料金情報を生成し、また、選択された「日付別フライト」および「チケットのクラス」に基づいて、特定のフライトおよび席のクラスで次の空席を計算します。
- スケジュールされたフライトの取り込み
  この Java エージェントは、スケジュール・ベースで実行されます。「表示状況」が「新規 (New)」となっている「利用可能なフライトのリスト」文書および「飛行機の座席情報」文書から収集した情報を使用して、「スケジュールされたフライト」文書を作成します。各フライトは、現在の日付より 1 カ月先まで、予定に組み込まれて行きます。
- スケジュールされたフライトの状況
  この Java エージェントは、スケジュール・ベースで実行されます。「利用可能なフライトのリスト」文書および「スケジュールされたフライト」文書から収集したフライトの日付と時刻に基づき、フライトの状況を、「新規 (new)」から「出発済み (departed)」に、また、「出発済み (departed)」から「到着済み (arrived)」に変更します。
- スケジュールされたフライトの更新
  この Java エージェントは、スケジュール・ベースで実行されます。フライトの状況をチェックし、このエージェントが前回実行されてから出発したフライトがあれば、そのフライトに関連したチケットを「処理されたチケット」ビューに移動することによって、チケットを更新します。
- チケット状況の更新
  この Java エージェントは、スケジュール・ベースで実行されます。「利用可能なフライトのリスト」文書および「スケジュールされたフライト」文書から収集したフライトの日付と時刻に基づき、チケットの状況を、「アクティブ (active)」から「処理済み (processed)」、または「非アクティブ (inactive)」に変更します。フライトがキャンセルされると、チケットは「非アクティブ (inactive)」になります。「処理済み (processed)」となっているチケットは、既に出発したフライトのチケットです。

「料金および座席番号生成」エージェント、および「固有都市コードのチェック」エージェントのソース・コードが、17 ページの『例: Lotus Domino エージェントのソース・コード』セクションに掲載されています。

アプリケーションの詳細

flights 社の従業員は、Lotus Domino Web インターフェースから、さまざまなタスクを行うことができます。これらのタスクは、顧客、フライト、フライト予約、および請求の、4 つの主なカテゴリーに分類されます。これらの各カテゴリーには、従業員が実行できるアクションが含まれます。実行できるアクションは、以下のとおりです。

- 顧客
  flights 社の従業員は、顧客の名前、パスワード、インターネット・アドレス、および自宅の住所を入力することで、顧客を追加することができます。入力された情報は、生成された顧客番号とともに Lotus Domino LDAP ディレクトリーに保管されます。この新規文書は、作成した後で、更新、削除、および表示することができます。
- フライト
  - flights 社の従業員は、出発および到着時刻、フライトの所要時間、出発都市と到着都市のコード、飛行機の機種、提供される食事の種類、スケジュールされたフライト日、およびファースト・クラスとエコノミー・クラスのチケットの料金を入力することで、フライトを追加できます。入力された情報は、「利用可能なフライトのリスト」フォームに保管されます。この文書は、作成した後で、表示または削除することができます。
  - また、flights 社の従業員は、市区町村、都道府県、国、および都市コードを入力することで、都市コードを追加することもできます。入力された情報は、都市コードが固有値であることが確認された後、「都市コード」フォームに保管されます。都市コードが固有値である場合は、文書が作成されます。この文書は、作成した後で、表示または削除することができます。
- フライトの予約
  flights 社の従業員は、スケジュールされた日付別フライト、チケットのクラス (エコノミーまたはファースト・クラス)、生成された料金、生成された座席番号、支払状況、顧客番号、および乗客情報を入力することで、フライトを予約できます。入力された情報は、「チケット情報」フォームに保管されます。この文書は、作成した後で、「アクティブ (active)」、「処理済み (processed)」、または「非アクティブ (inactive)」のいずれかの状況で表示されます。
- 請求
  flights 社の従業員は、クレジット・カードの種類、番号、有効期限、および顧客番号を入力することで、チケットについての請求書を生成できます。入力された情報は、「クレジット情報」フォームに保管されます。この文書は、作成した後で、表示または更新できます。

Lotus(R) Domino(TM) 環境の主要な発見事項

flights シナリオの Lotus Domino 環境の作成および使用の過程で判明した、主要な発見事項について、以下にリストします。

- 一部の flights 文書では、固有値の生成にエージェントを使用しました。このエージェントは、WebQueryOpen イベントによって呼び出されました。このエージェントは、新規文書を作成する際に値を生成しましたが、生成された値は、文書の保管時に保管されませんでした。 Lotus Domino Designer ヘルプ・テキストからの以下の抜粋が、WebQueryOpen イベントがどのように機能したかについて、および、この値が保管されなかった理由について説明しています。

  “WebQueryOpen エージェントは、ユーザーがフォームまたは文書をオープンするときに実行されますが、ユーザーが文書を保管するときには実行されません。つまり、WebQueryOpen エージェントによって設定された、計算結果を含むフィールドは、ユーザーが文書をサブミットしたときには保管されませ

ん。計算結果を含むフィールドが保管されるようにするには、 WebQuerySave エージェントでフィールドを再計算するか、フォーム・プロパティー「全フィールドに対して HTML を生成 (Generate HTML for all fields)」を設定することができます。”

フォーム・プロパティー「全フィールドに対して HTML を生成 (Generate HTML for all fields)」を選択すると、計算された値は、文書内の他のすべてのフィールドと一緒に保管されました。

- Web を介して文書をサブミットした後に、デフォルトのテキストである「フォームは処理されました (Form processed)」の代わりにビューを表示させるには、以下の手順を実行します。
  - 表示用に計算される、非表示のテキスト・フィールドをフォームに作成します。
  - フィールドに、$$Return という名前を付けます。
  - フィールド値には、次のような数式を使用します。
    “[/”+@Subset(@DbName;-1)+“/YourViewName?OpenView&DblClkTarget=_self]”

    使用するビューの名前にスペースが含まれている場合は、スペースの代わりに「+」を使用してください。 (例: Your+View+Name)
    使用するビューの名前が、他のビューの下にカテゴリー化されている場合には、二重の円記号 (¥¥) を使用してください。 (例: YourView¥¥ViewName)
- 「ビューのプロパティー (view properties)」の「アドバンスト (advanced)」タブで、「Web アクセス用 (For Web Access)」の下の「ブラウザーでアプレットを使用 (Use applet in the browser)」ビュー・プロパティーを選択すると、パフォーマンスが低下し、また、その他の予期しない動作が発生しました。

- 「フライト乗客」フォームで、 Web 上でのデータ入力時にフォーム内にデータが保持されるようにするには、フォーム・プロパティー「全フィールドに対して HTML を生成 (Generate HTML for all fields)」を選択する必要がありました。このフォーム・プロパティーが選択されていないと、顧客フォームの次のタブをクリックしたとき、他のタブ内のデータは保持されません。このフォーム・プロパティーを選択していると、それぞれのタブが選択された際にもデータは保持されました。

- Web から新規の固有都市コードを作成する際には、固有の都市コードのみが保管されるようにすることを求めました。

以下に示す Lotus Domino Designer のヘルプ・テキストからの抜粋は、固有値であるかどうかのチェックを行う際に WebQuerySave イベントがどのように機能するかについて説明しています。

“WebQuerySave イベントをプログラムし、値が「0」の SaveOptions フィールドをフォームに追加することにより、ユーザーの入力したデータに対して実行する CGI プログラムをシミュレートします。エージェントを実行すると、新規 Notes(TM) 文書を生成することなく、入力済みのフォームからフィールド値を収集することができます。”

SaveOptions を 0 に設定すると、文書が保管されないようにすることができます。この値が 1 に設定さ

れるとすぐに、文書は保管されます。サブミット・ボタン内の SaveOptions に対して if 文でチェックを行う必要はありません。 Lotus Domino は、SaveOptions フィールドの値に基づいて保管を処理します。

例えば、このシナリオで使用した解決策は以下のとおりです。

1. フォーム内に、サブミット・ボタンを作成し、また、SaveOptions という名前の隠しフィールドを、初期値を 0 に設定して作成する。
2. サブミット・ボタンに、次の式をコーディングする。
   @Command([FileSave]);
   @Command([FileCloseWindow])
3. WebQuerySave イベントで、エージェントを実行する式を入力する。
4. エージェントに、以下のコードを追加する。
   If (foundCityCode) Then
   Print “<SCRIPT LANGUAGE=JavaScript(TM)>”
   Print “alert(””Duplicate City Code found. Please enter a new city code.””)”
   Print “location.href = ””../../” + file + “/City+Codes?OpenForm””””
   Print “</SCRIPT>”
   Else
   ’// The following line will set the SaveOptions on the document as 1 which will cause Notes to save the document.
   Set item = doc.ReplaceItemValue(“SaveOptions”, 1)
   End If

参照

- 「Lotus Domino Release 5.0: A Developer’s Handbook」 IBM Redbook(R) (SG24-5331-01)
- IBM Lotus Domino for iSeries(TM) - OS/400(R) Web サイト
  http://www.ibm.com/servers/eserver/iseries/domino/
- IBM Lotus Domino for iSeries (PartnerWorld(R) for Developers)
  http://www.as400.ibm.com/developer/domino/
- Lotus Web サイト
  http://www.lotus.com

例: Lotus Domino エージェントのソース・コード

このセクションには、flights アプリケーションの Lotus(R) Domino(TM) エージェントのうち、2 つのエージェントのソース・コードが含まれています。最初の例は、Java(TM) で記述された「料金および座席番号生成」エージェントです。このエージェントは、料金情報の生成と、次の空席の計算を行います。 2 番目の例は、LotusScript で記述された「固有都市コードのチェック」エージェントです。このエージェントは、都市コードが、固有の値であるかどうかを検証します。

例 1:「料金および座席番号生成」エージェント

```
////////////////////////////////////////////////////////////////////////
/*
* This Java Agent will run based on the user clicking an action button.
* It will generate the price information and calculate the next available seat
* based on the FlightByDate and Ticket Class selected.
```

```
*
* Scenario Name: TFC Flights
*
* Java Version: JDK 1.1.8
*/
////////////////////////////////////////////////////////////////////////
```

```
import lotus.domino.*;
```

```
public class JavaAgent extends AgentBase {
```

```
public void NotesMain() {
```

```
System.out.println("Starting: Generate Price and Seat Number agent.");
```

```
try
{
Session session = getSession();
AgentContext agentContext = session.getAgentContext();
```

```
generateSeat(session, agentContext);
```

```
System.out.println("Done: Generate Price and Seat Number agent.");
}
catch(Exception e) {
e.printStackTrace();
}
```

```
} // end Notes(TM) main
```

```
public void generateSeat(Session session, AgentContext agentContext)
{
// get the Flight by date and ticket class data from the current document
boolean isFirstClass = false; //initially set this flag to false (i.e. coach)
```

```
try
{
Database db = agentContext.getCurrentDatabase();
DocumentCollection collection = agentContext.getUnprocessedDocuments();
if (collection.getCount() < 1)
{
// there was a problem accessing the current doc, quit agent
System.out.println("Error with the current document, agent ending.");
return;
} // end if
else
{
Document doc = collection.getFirstDocument();
String flightByDate = doc.getItemValueString("FlightByDate");
String ticketClass = doc.getItemValueString("TicketClass");
DocumentCollection scheduledFlightDC = db.search("SELECT ((Form = ¥"Scheduled Flights¥")) &
((@Contains(ScheduledFlightByDate; ¥"" + flightByDate + "¥")))");
if (scheduledFlightDC.getCount() < 1)
{
// invalid flight by date specified - no matching flight by date found
System.out.println("Invalid flight by date specified - no matching flight by date found, agent ending.");
return;
} // end if
else
{
int seatsRemaining = 0;
int seatNumber = 0;
Document scheduledFlightDoc = scheduledFlightDC.getFirstDocument();



if (ticketClass.equals("First Class"))
{
isFirstClass = true;



seatsRemaining = scheduledFlightDoc.getItemValueInteger("FirstClassSeatsAvailable");



//check to see if a seat is still available
if (seatsRemaining >= 1)
{
// get the next available seat number
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString("ScheduledAirplaneType"));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;
```

```
String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue("SeatNumber", strObj);


int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);


// a seat is available
Integer intObject = new Integer(seatsRemaining-1); //decrement num of available seats
scheduledFlightDoc.replaceItemValue("FirstClassSeatsAvailable", intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1


else
{ //if seats remaining = 0, check to see if any tickets have been deleted
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedFirstClassSeats");
if (deletedSeats != null)
{ // there's a cancelled seat(s) available, book the seat
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// update the deleted seats, removing the seat that was just booked and leaving the remaining as is
scheduledFlightDoc.replaceItemValue("DeletedFirstClassSeats", deletedSeats.substring(index + 1));
scheduledFlightDoc.save(true, true);


String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);


int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);


} // end if deletedSeats != null
else
{
// the flight is full
```

```
System.out.println ("<SCRIPT LANGUAGE=JavaScript(TM)>");
System.out.println ("alert(¥"No more seats are available on this flight. Please select a new flight.¥")");
System.out.println ("</SCRIPT>");
} // end else


} // end seats remaining = 0


} // end if first class
else
{ // the ticket is for a coach seat
seatsRemaining = scheduledFlightDoc.getItemValueInteger("CoachSeatsAvailable");


//check to see if a seat is still available
if (seatsRemaining >= 1)
{
// get the next available seat number
seatNumber = getNextAvailableSeat(db, isFirstClass,
scheduledFlightDoc.getItemValueString("ScheduledAirplaneType"));
int nextAvailableSeat = seatNumber - seatsRemaining + 1;


String strObj = String.valueOf(nextAvailableSeat);
doc.replaceItemValue("SeatNumber", strObj);


int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);


// a seat is available
Integer intObject = new Integer(seatsRemaining-1); //decrement num of available seats
scheduledFlightDoc.replaceItemValue("CoachSeatsAvailable", intObject);
scheduledFlightDoc.save(true, true);
} //end if seats remaining >= 1


else
{ //if seats remaining = 0, check to see if any tickets have been deleted
String deletedSeats = scheduledFlightDoc.getItemValueString("DeletedCoachSeats");
if (deletedSeats != null)
```

```
{ // there's a cancelled seat(s) available, book the seat
int index = deletedSeats.indexOf(";");
String availSeat = deletedSeats.substring(0, index);
Integer intObject = Integer.valueOf(availSeat);
// update the deleted seats, removing the seat that was just booked and leaving the remaining as is
scheduledFlightDoc.replaceItemValue("DeletedCoachSeats", deletedSeats.substring(index + 1));
scheduledFlightDoc.save(true, true);


String strObj = String.valueOf(availSeat);
doc.replaceItemValue("SeatNumber", strObj);


int price = getPrice(flightByDate, session, agentContext, isFirstClass); //get the price info for this ticket
Integer priceInt = new Integer(price);
doc.replaceItemValue("TicketPrice", priceInt);
doc.save(true, true);


} // end if deletedSeats != null
else
{
// the flight is full
System.out.println ("<SCRIPT LANGUAGE=JavaScript>");
System.out.println ("alert(¥"No more seats are available on this flight. Please select a new flight.¥")");
System.out.println ("</SCRIPT>");
}
} // end seats remaining = 0
} // end else ticket is coach
}
} // end else
} // end try


catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}


} // end method generateSeat
```

```
public int getNextAvailableSeat(Database db, boolean isFirstClass, String airplaneType)
{
// go to Airplane Seat Info view and find the matching plane type
int seatNum = 0;
try
{
DocumentCollection seatDC = db.search("SELECT ((Form = ¥"Airplane Seat Information¥")) &
((@Contains(SeatAirplaneType; ¥"" + airplaneType + "¥")))");
if (seatDC.getCount() < 1)
{
// no matching airplane type found
System.out.println("Invalid airplane model specified - no matching record found, agent ending.");
return 0;
} // end if
else
{
Document seatDoc = seatDC.getFirstDocument();
if (isFirstClass)
{
seatNum = seatDoc.getItemValueInteger("FirstClassSeats");
}
else
{
seatNum = seatDoc.getItemValueInteger("CoachSeats");
}



} // end else
} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}



return seatNum;
} // end method getNextAvailableSeat



public int getPrice(String flightByDate, Session session, AgentContext agentContext, boolean isFirstClass)
{
int price = 0;
try
{
```

```
// calculate flight number
int index = flightByDate.lastIndexOf("_");
String flightNumber = flightByDate.substring(0, index);



Database db = agentContext.getCurrentDatabase();
DocumentCollection flightDC = db.search("SELECT ((Form = ¥"List Of Available Flights¥")) &
((@Contains(AvailableFlightNumber; ¥"" + flightNumber + "¥")))");



if (flightDC.getCount() < 1)
{
// there was a problem accessing the current doc, quit agent
System.out.println("Error with the current document, agent ending.");
return 0;
} // end if
else
{
Document doc = flightDC.getFirstDocument();
if (isFirstClass)
{
price = doc.getItemValueInteger("FirstClassPrice");
}
else
{
price = doc.getItemValueInteger("CoachPrice");
}



} // end else



} // end try
catch (NotesException ne) {
ne.printStackTrace();
}
catch (Exception e) {
e.printStackTrace();
}



return price;
} // end method generateSeat
} // end class
```

例 2:「固有都市コードのチェック」エージェント

```
Sub Initialize

'//===================================== '// This Domino agent receives a City Code as input
from a browser using the City Codes Form.
'// This agent uses that information along with information from the existing City Code documents (created
from the City Codes form).
'// This agent uses the veiw 'City Codes' to search through all the City Code documents. If a match is
found, the user is
'// notified and allowed to create a new City Code document. If no match is found, the City Code document
will be created.
'//=====================================

'// Set foundCityCode variant
Dim foundCityCode As Variant
foundCityCode = False

'// Create a notes session
Dim session As New NotesSession

Messagebox "Running agent " & session.CurrentAgent.Name & " in database " &
session.CurrentDatabase.Title & " as " & session.CommonUserName

'// Open this database
Dim db As NotesDatabase
Set db = session.CurrentDatabase
If Not ( db.IsOpen) Then
Messagebox "Database " & session.CurrentDatabase.Title & " did not open. Agent ending."
Exit Sub
End If

'// Set doc equal to the current session transient values (i.e. the city code document code variables)
Dim doc As NotesDocument
Set doc = session.DocumentContext
Dim unid As String
Dim file As String
file = db.FileName
```

```
'// Set the view to the City Codes view, to locate a City Codes document
Dim CCview As NotesView
Dim CCdoc As NotesDocument
Set CCview = db.GetView("City Codes")
Set CCdoc = CCview.GetFirstDocument


'// Search through the City Code documents until a match is found or no match is found
While ((Not (CCdoc Is Nothing)) And (Not foundCityCode))
If (CCdoc.Code(0) = doc.Code(0)) Then
unid = CCdoc.UniversalID
foundCityCode = True
Else
Set CCdoc = CCview.GetNextDocument (CCdoc)
End If
Wend


'// If match found, display a message and allow user to create a new City Code document
'// If no match found, allow user to create a new City Code document
If (foundCityCode) Then
Print "<SCRIPT LANGUAGE=JavaScript>"
Print "alert(""Duplicate City Code found. Please enter a new city code."")"
Print "location.href = ""../../" + file + "/City+Codes?OpenForm"""
Print "</SCRIPT>"
Else
On Error Goto Errhandle
'// Call doc.Save(True, True)
'// Print "<SCRIPT LANGUAGE=JavaScript>"
'// Print "location.href = ""../../" + file + "/City+Codes/doc?SaveDocument"""
'// Print "alert(""Saving."")"
'// Print "location.href = ""../../" + file + "/City+Codes?OpenView"""
'// Print "</SCRIPT>"
Set item = doc.ReplaceItemValue("SaveOptions", 1)
End If


Errhandle:
' Use the Err function to return the error number and
' the Error$ function to return the error message.
Messagebox "Error" & Str(Err) & ": " & Error$
Exit Sub


End Sub
```

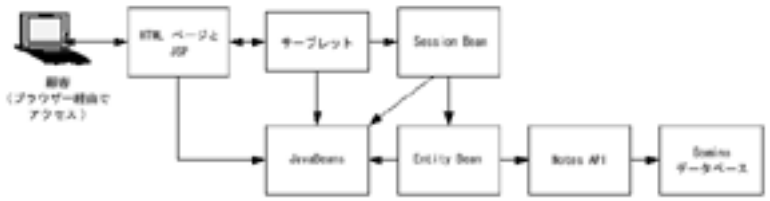
第 3 章 WebSphere(R) Application Server 環境の概要

WebSphere Application Server 環境は、iSeries(TM) システム・テストの flights シナリオの作業のフェーズ 2 でセットアップされました。 flights 社は、フェーズ 2 において、顧客が同社の Web サイトからフライトを予約できるようにすることを目標としていました。 同社では、このインターフェースの構築に、HTML ページ、Java(TM) Server Pages (JSP)、JavaBeans(TM)、Enterprise Bean、およびサーブレットを使用することを決定しました。このインターフェースを使用することで、同社の顧客は、フライトの表示と予約、フライト情報の検索、顧客情報の更新、および決済をオンラインで実行できるようになりました。

アプリケーション・モデル

flights アプリケーションでは、図 1 に示したアプリケーション・モデルを使用します。このモデルでは、ブラウザーは、Enterprise Bean を使用してビジネス・ロジックと対話するサーブレットを介して、間接的に Java Server Pages (JSP) にアクセスします。 Enterprise Bean は、必要な情報を Lotus(R) Domino(TM) データベースから抽出します。サーブレットは、クライアント要求を受信すると、必要な計算を実行して JavaBeans を作成します。 JSP が、対応する JavaBeans によって呼び出されます。JSP は、必要な情報を JavaBeans から抽出して、HTML ページにマージします。次に、ブラウザーが HTML をインタープリットして表示します。

図 1 flights アプリケーションのモデル



第 4 章 アプリケーション・プロセスのフロー

flights アプリケーションでは、多数の HTML ページと JSP ページが使用されます。表 1 に HTML ページのリストを、表 2 に JSP のリストを示します。アプリケーションのフローを、図 2 に示します。

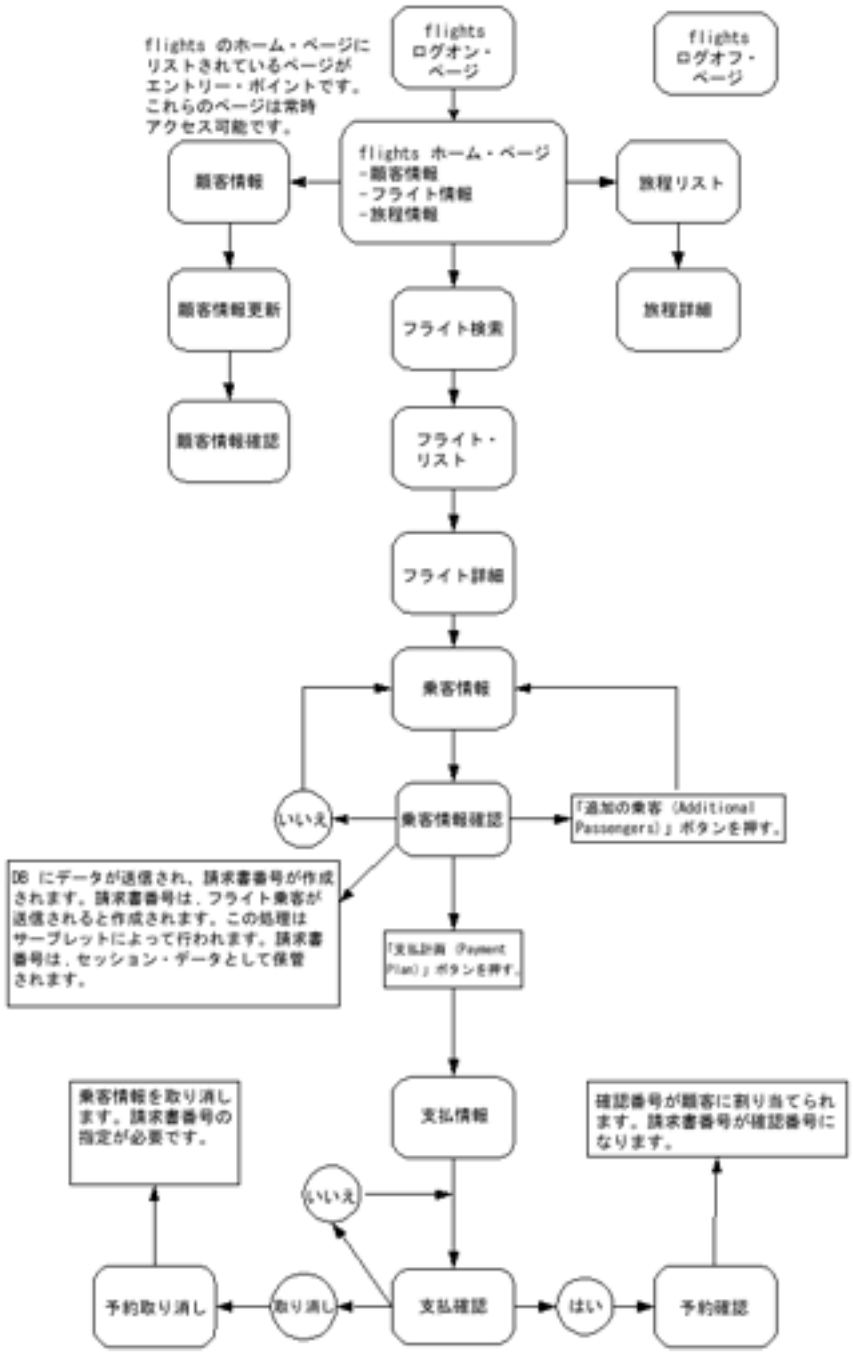
表 1 flights アプリケーションの HTML ページ

HTML ページ	機能
homePage	顧客は、以下の情報を表示するよう選択することができる • フライト情報 • 顧客情報 • 旅程情報
index	menu、top、および homePage の各フレームをまとめる
logOn	顧客がログオンできるようにする
menu	左側のメニュー・オプションを提供する
top	ロゴを含むページのトップを提供する

表 2 フライト・アプリケーションの JSP

JSP	機能
customerInformation	顧客に関する情報を表示する
logOff	顧客がログオフできるようにする
customerInformationUpdate	顧客が顧客情報を更新できるようにする
customerInformationConfirmation	顧客が自ら更新した顧客情報を確認できるようにする
flightsSearch	フライトを検索するための基準のリストを提供する
flightsList	検索基準に一致するフライトのリストを提供する
flightsDetails	特定のフライトに関する詳細なフライト情報を提供する
passengerInformation	顧客が乗客情報を入力できるようにする
passengerInformationVerification	顧客が乗客情報を確認できるようにする
paymentInformation	顧客が支払情報を入力できるようにする
paymentInformationVerification	顧客が支払情報を確認できるようにする
reservationCancel	顧客が予約の取り消しを行えるようにする
bookConfirmation	予約を確認する
itineraryList	顧客の旅程のリストを提供する
itineraryDetails	特定の旅程についての詳細な旅程情報を提供する

図 2 フライト・アプリケーションのフロー



開発環境

このアプリケーションの開発時には、以下の製品が使用されました。

- WebSphere Studio
- Visual Age for Java Enterprise Edition
- WebSphere Studio Application Developer

当初、JSP は WebSphere Studio を使用して作成され、Enterprise Bean は Visual Age for Java を使用して作成されました。開発の途中で、WebSphere Application Server バージョン 4.0 へ移行しました。この変更に伴い、JSP および Enterprise Bean の開発環境が WebSphere Studio Application Developer へと移行されました。

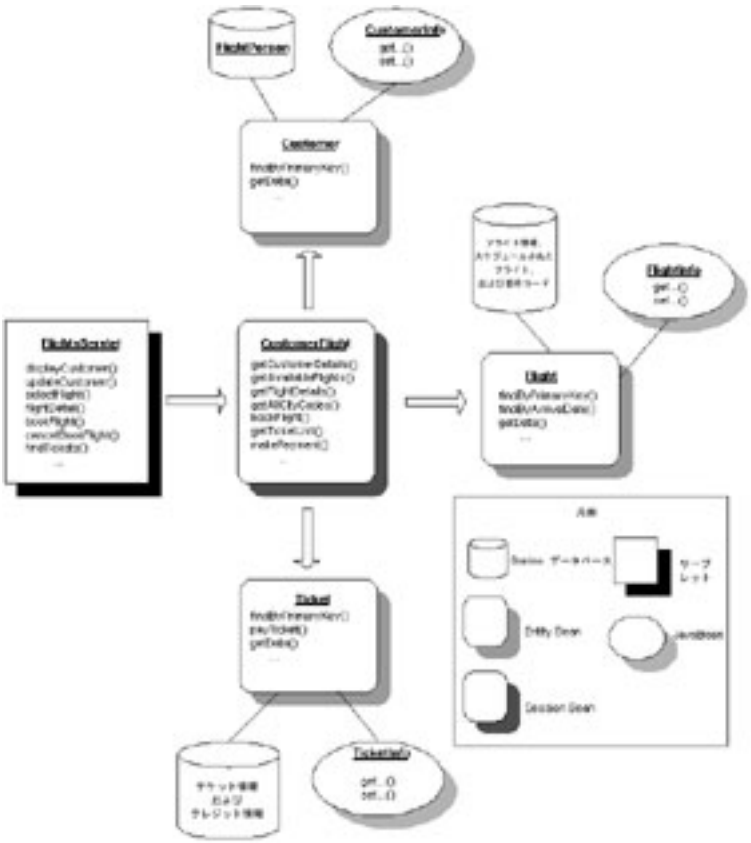
WebSphere(R) Application Server 環境のアプリケーション・フロー

アプリケーションの詳細

このセクションでは、サーブレット、JavaBeans(TM)、Enterprise Bean、および JSP の使用を含む、 flights アプリケーションのアプリケーション・モデルの詳細について説明します。 FlightsServlet という 1 つのサーブレットがあり、これがメイン・コントローラーとして機能します。すべての要求はこのサーブレットに送信され、要求されたタスクが実行されます。このサーブレットは、大部分の処理を、Session Bean である CustomerFlight にアクセスして実行します。 CustomerFlight Session Bean では Entity Bean (Customer、Flight、および Ticket) を使用して、要求されたタスクを実行します。

図 1 に、サーブレットと Enterprise Bean との間の関係を示します。

図 1 アプリケーションの関係



設計上の考慮事項

当初、Enterprise Bean 内で Lotus(R) Domino(TM) JDBC ドライバーを使用して、Lotus Domino データベースにアクセスする計画でした。しかし、WebSphere Application Server では、iSeries(TM) 上で Lotus Domino JDBC ドライバーを使用するデータ・ソースを作成できません。また、iSeries では Lotus Domino JDBC ドライバーがまだ移植されていないため、独自の接続プールをインプリメントすることもできませんでした。 iSeries 対応の Lotus Domino JDBC ドライバーは、将来のリリースで提供される予定です。上記の制限のため、Entity Bean 内で Lotus Domino API を使用して、Lotus Domino データベースにアクセスすることとしました。

アプリケーション設計時に、Bean 管理パーシスタンスの Entity Bean を使用することを決定しました。Entity Bean が Lotus Domino API を使用して Lotus Domino データベースにアクセスすることから、Bean 管理の Entity Bean を使用する必要がありました。

FlightsServlet

flights アプリケーションでは、FlightsServlet を使用してアプリケーションのフローを制御します。FlightsServlet は、flights 社の顧客に対して、顧客情報を表示および更新する機能、スケジュールされたフライトのデータを表示する機能、および Web ブラウザーでフライトを予約する機能を提供するために使用されます。このサーブレットの呼び出す JSP ページが、データの表示に使用されます。これらの JSP ページは、最小限の処理作業を行います。処理作業の大部分は、FlightsServlet によって、以下のメソッドを使用して実行されます。

- doPost
  - Web サーバーと、要求を行うブラウザーとの間に、セッションを作成します。
  - 各要求を調べ、jspRequest の値に基づいてサーブレット内の適切なメソッドへ送ります。
- doGet
  - Web サーバーと、要求を行うブラウザーとの間に、セッションを作成します。
  - jspRequest の値を適宜設定し doPost() を呼び出します。
- errorHandle
  - サーブレットまたは Enterprise Bean で発生した例外を処理します。
  - 発生したエラーに関するメッセージをユーザーに表示します。
- init
  - サーバーがサーブレットのインスタンスを作成した直後にサーバーによって呼び出されます。
  - CustomerFlight のホーム・オブジェクトを作成して、探索します。
- bookFlight
  - CustomerFlight Enterprise Bean を使用して、指定された顧客のフライトを予約します。
  - passengerInformation.jsp を表示します。
- cancelBookFlight
  - CustomerFlight Enterprise Bean を使用して、特定の顧客が予約していたフライトをキャンセルします。
  - reservationCancel.jsp を表示します。
- confirmFlight
  - CustomerFlight Enterprise Bean を使用して、予約されたフライトを確認し、決済します。

  - bookConfirmation.jsp を表示します。
- displayCustomer
  - CustomerFlight Enterprise Bean を使用して、指定された顧客に関する情報を取得します。
  - customerInformation.jsp を表示します。
- displayJSP
  - JSP ページを呼び出します。JSP が必要とするデータはすべて、セッションに含まれています。
- findTickets
  - CustomerFlight Enterprise Bean を使用して、チケットのリストを取得します。
  - itineraryList.jsp を表示します。
- flightDetail
  - CustomerFlight Enterprise Bean を使用して、詳細なフライト情報を取得します。
  - flightsDetails.jsp を表示します。
- flightList
  - CustomerFlight Enterprise Bean を使用して、全都市コードのリストを取得します。
  - flightSearch.jsp を表示します。
- getCustomerFlight
  - CustomerFlightHome クラスを探索することによって、CustomerFlightHome オブジェクトを作成します。
- getInitialContext
  - 指定された URL の初期コンテキストを取得します。
- itineraryDetails
  - CustomerFlight Enterprise Bean を使用し、旅程番号と日付別フライトに基づいて、旅程とフライトの詳細を取得します。
  - itineraryDetails.jsp を表示します。
- selectFlight
  - CustomerFlight Enterprise Bean を使用し、指定された検索条件に基づいて、全フライトのリストを取得します。
  - flightsList.jsp を表示します。
- updateCustomer
  - CustomerFlight Enterprise Bean を使用して、顧客データを更新します。
  - customerInfoConfirmation.jsp を表示します。

Enterprise Bean

flights アプリケーションでは、FlightsServlet はメイン・コントローラーとして機能します。すべての要求はこのサーブレットに送信され、要求されたタスクが実行されます。このサーブレットは、大部分の処理を、Session Bean である CustomerFlight にアクセスして実行します。 CustomerFlight Session Bean では Entity Bean (Customer、Flight、および Ticket) を使用して、要求されたタスクを実行します。

CustomerFlight Enterprise Bean

CustomerFlight Enterprise Bean は、ステートレス Session Bean です。 CustomerFlight Enterprise Bean は、顧客が行う次のようなタスクを実行するためのメソッドを提供します。

- 顧客情報の表示または更新
- 利用可能なフライトのリストの取得
- 旅程の詳細の表示
- フライトの予約
- 予約したフライトの決済

CustomerFlight Session Bean のメソッドについて、以下に説明します。

- bookFlight() メソッドは、予約されたチケットの情報を含む TicketInfo オブジェクトを戻します。以下のタスクが実行されます。
  1. 請求書番号が指定されているかどうかをチェックする。指定されていない場合は、新しい請求書番号を使用して新規のチケットを作成します。請求書番号が指定されている場合は、指定された請求書番号を使用して新規のチケットを作成します。チケットは、Ticket Enterprise Bean を使用して作成されます。
  2. Ticket Enterprise Bean 内で getData() メソッドが呼び出され、データが TicketInfo オブジェクトに保管される。
  3. TicketInfo オブジェクトが戻される。
- cancelBookFlight() メソッドは、指定された請求書番号のチケットをキャンセルします。以下のタスクが実行されます。
  1. Ticket Enterprise Bean の findByInvoiceNumber() メソッドを使用して、指定された請求書番号を持つチケットのリストを取得する。
  2. 次に、チケットのリストが順次処理され、各チケットに対して remove() メソッドが呼び出されて、Lotus Domino データベースから文書が除去される。
- getAllCityCodes() メソッドは、「都市コード」文書にあるすべての都市コードのリストを取得します。以下のタスクが実行されます。
  1. このメソッドが「都市コード」フォームからすべての都市コードを選択し、その情報が FlightInfo オブジェクトに保管される。
  2. FlightInfo オブジェクトが戻される。
- getAvailableFlights() メソッドは、指定された検索条件に一致するすべてのフライトのリストを含む、FlightInfo オブジェクトを戻します。以下のタスクが実行されます。
  1. パラメーター値がチェックされ、Flight Enterprise Bean を使用して、検索条件に基づいた検索が実行される。
  2. Flight Enterprise Bean 内の getData() メソッドが呼び出され、データが FlightInfo オブジェクトに保管される。
  3. FlightInfo オブジェクトが戻される。
- getConnection() メソッドは、Lotus Domino API を利用して、Lotus Domino データベースへの接続を戻します。以下のタスクが実行されます。
  1. 環境変数を使用して、接続が作成される。
  2. Lotus Domino データベースへの新規セッションが作成される。
  3. データベース・オブジェクトが作成されて、戻される。
- getCustomerDetails() メソッドは、指定された顧客番号の顧客情報を含む、 CustomerInfo オブジェクトを戻します。以下のタスクが実行されます。
  1. Customer Enterprise Bean を使用して、顧客番号に基づく主キーの検索が実行される。

2. Customer Enterprise Bean 内で getData() メソッドが呼び出され、データが CustomerInfo オブジェクトに入れられて戻される。

- getFlightDetails() メソッドは、指定された日付別フライトのフライト情報を含む、FlightInfo オブジェクトを戻します。以下のタスクが実行されます。
  1. Flight Enterprise Bean を使用して、日付別フライトに基づく主キーの検索が実行される。
  2. Flight Enterprise Bean 内で getData() メソッドが呼び出され、データが FlightInfo オブジェクトに入れられて戻される。
- getInitialContext() メソッドは、Entity Bean を作成するための初期コンテキストを戻します。
- getTicketList() メソッドは、顧客番号または請求書番号に基づき、すべてのチケットのリストを含む TicketInfo オブジェクトを戻します。以下のタスクが実行されます。
  1. Ticket Enterprise Bean を使用して、渡されたパラメーターに基づき、顧客番号または請求書番号による検索が実行される。
  2. Ticket Enterprise Bean 内で getData() メソッドが呼び出され、データがベクターに格納される。
  3. ベクターが TicketInfo オブジェクトに格納される。
  4. TicketInfo オブジェクトが戻される。
- makePayment() メソッドは、TicketInfo オブジェクトを戻します。このメソッドは、予約されたフライトの決済に使用されます。このメソッドは、同じ請求書番号のすべてのチケットを更新します。以下のタスクが実行されます。
  1. Ticket Enterprise Bean を使用して、請求書番号に基づき、請求書番号による検索が実行される。
  2. Ticket Enterprise Bean 内で payTicket() メソッドが呼び出される。
  3. Ticket Enterprise Bean 内で getData() メソッドが呼び出され、データが TicketInfo オブジェクトに入れられて戻される。
- setCustomerHome() メソッドは、CustomerHome クラスを探索することによって、CustomerHome オブジェクトを作成します。
- setFlightHome() メソッドは、FlightHome クラスを探索することによって、FlightHome オブジェクトを作成します。
- setTicketHome() メソッドは、TicketHome クラスを探索することによって、TicketHome オブジェクトを作成します。
- updateCustomer() メソッドは、顧客の個人情報を更新します。以下のタスクが実行されます。
  1. Customer Enterprise Bean を使用して、顧客番号に基づく主キーの検索が実行される。
  2. Customer Enterprise Bean 内で、以下のメソッドが呼び出される。
     - setCustomerFirstName() メソッド
     - setCustomerMiddleInitial() メソッド
     - setCustomerLastName() メソッド
     - setCustomerAddress() メソッド
     - setCustomerCity() メソッド
     - setCustomerState() メソッド
     - setCustomerZipCode() メソッド
     - setCustomerCountry() メソッド
     - setCustomerPhoneNumber() メソッド
     - setCustomerInernetAddress() メソッド
     - CustomerInfo オブジェクトにデータを格納する、getData() メソッド

3. CustomerInfo オブジェクトが戻される。

Customer Bean

Customer Enterprise Bean は、flights 社の顧客を表すために使用される Entity Bean です。この Bean は、顧客データに対して実行される、次のようなタスクを実行するためのメソッドを提供します。

- 顧客情報の表示
- 顧客情報の更新

Customer Enterprise Bean は、Bean 管理パーシスタンスを使用し、 names.nsf Lotus Domino データベース内の FlightPerson フォームにマップされます。 FlightPerson フォームのレイアウトは、 6 ページの『Lotus(R) Domino(TM) 環境のワークフロー』セクションの表 10 に示されています。
Customer Enterprise Bean には、ejbFindBy メソッドが 1 つ含まれています。この finder メソッドは、ホーム・インターフェースで定義されています。表 1 に、この finder メソッドを示します。

表 1 Customer Enterprise Bean の finder メソッド

メソッド名	Select 文	説明
ejbFindByPrimaryKey()	“SELECT (Form = ¥”FlightPerson¥“) & (PersonalID = ” + customerNumber.trim() +“)”	顧客番号に基づいて顧客を検索

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 2 に示します。

表 2 Customer の getter および setter メソッド

getter メソッド	setter メソッド	フォームのフィールド名	データ型
getCustomerAddress()	setCustomerAddress()	FlightPerson: StreetAddress	String
getCustomerCity()	setCustomerCity()	FlightPerson: City	String
getCustomerCountry()	setCustomerCountry()	FlightPerson: Country	String
getCustomerFirstName()	setCustomerFirstName()	FlightPerson: FirstName	String
getCustomerInternetAddress()	setCustomerInternetAddress()	FlightPerson: InternetAddress	String
getCustomerLastName()	setCustomerLastName()	FlightPerson: LastName	String
getCustomerMiddleInitial()	setCustomerMiddleInitial()	FlightPerson: MiddleInitial	String
getCustomerNumber()	setCustomerNumber()	FlightPerson: PersonalID	String
getCustomerPhoneNumber()	setCustomerPhoneNumber()	FlightPerson: PhoneNumber	String
getCustomerState()	setCustomerState()	FlightPerson: State	String
getCustomerZipCode()	setCustomerZipCode()	FlightPerson: Zip	String
getData()		Enterprise Bean 内の値を収容した CustomerInfo JavaBean	CustomerInfo

ejbLoad() メソッドは、特定の FlightPerson 文書からデータを取得して Bean プロパティーに入れるために使用されます。

ejbStore() メソッドは、Bean プロパティーから、特定の FlightPerson 文書に含まれるデータを更新するために使用されます。

CustomerInfo

CustomerInfo は、顧客情報を格納するために使用される JavaBean です。この JavaBean は適切な JSP に渡されます。JSP はこの JavaBean を使用して、特定の顧客データを取得します。

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 3 に示します。

表 3 CustomerInfo の getter および setter メソッド

getter メソッド	setter メソッド	値	データ型
getCustomerAddress()	setCustomerAddress()	顧客の住所	String
getCustomerCity()	setCustomerCity()	顧客の市区町村	String
getCustomerCountry()	setCustomerCountry()	顧客の国	String
getCustomerFirstName()	setCustomerFirstName()	顧客のファーストネーム	String
getCustomerInternetAddress()	setCustomerInternetAddress()	顧客のインターネット・アドレス	String
getCustomerLastName()	setCustomerLastName()	顧客のラストネーム	String
getCustomerMiddleInitial()	setCustomerMiddleInitial()	顧客のミドル・ネームのイニシャル	String
getCustomerPhoneNumber()	setCustomerPhoneNumber()	顧客の電話番号	String
getCustomerState()	setCustomerState()	顧客の都道府県	String
getCustomerZipCode()	setCustomerZipCode()	顧客の郵便番号	String

Customer Bean および CustomerInfo Bean で使用されているソース・コードは、63 ページの『例 : Customer Bean』セクションに示されています。

Flight Bean

Flight Enterprise Bean は、フライトを表すために使用される Entity Bean です。この Bean は、フライト・データに対して実行される、次のようなタスクを実行するためのメソッドを提供します。

- 特定の検索条件に基づいた、利用可能なフライトのリストの取得
- 日付別フライトの値に基づいた、特定のフライト情報の取得

Flight Enterprise Bean は、Bean 管理パーシスタンスを使用し、 flights.nsf Lotus Domino データベース内の「利用可能なフライトのリスト」フォームおよび「スケジュールされたフライト」フォームにマップされます。「利用可能なフライトのリスト」フォームのレイアウトは 6 ページの『Lotus(R) Domino(TM) 環境のワークフロー』セクションの表 2 に示されており、「スケジュールされたフライト」フォームのレイアウトは同セクションの表 3 に示されています。

Flight Enterprise Bean は、複数の ejbFindBy メソッドを含んでいます。これらの finder メソッドは、ホーム・インターフェースで定義されています。表 4 に各 finder メソッドを示します。ここで、メソッド名の先頭には ejbFindBy が付きます。

表 4 Flight の finder メソッド

メソッド名	Select 文	説明
ArrivalCity()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0) ｜ (FirstClassSeatsAvailable>0))”	到着都市に基づいてフライトを検索
ArrivalCityArrivalDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0) ｜ (FirstClassSeatsAvailable>0))”	到着都市と到着日に基づいてフライトを検索
ArrivalCityArrivalDate DepartureCity()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0) ｜ (FirstClassSeatsAvailable>0))”	到着都市、到着日、および出発都市に基づいてフライトを検索

ArrivalCityArrivalDate DepartureCityDepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着都市、到着日、出発都市、および出発日に基づいてフライトを検索
ArrivalCityArrivalDate DepartureCitySeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、到着日、出発都市、出発日、および席のタイプに基づいてフライトを検索

ArrivalCityArrivalDate DepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着都市、到着日、および出発日に基づいてフライトを検索
ArrivalCityArrivalDate DepartureDateSeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、到着日、出発日、および席のタイプに基づいてフライトを検索

ArrivalCityArrivalDate SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、到着日、および席のタイプに基づいてフライトを検索
ArrivalCityDepartureCity()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0) ｜ (FirstClassSeatsAvailable>0))”	到着都市と出発都市に基づいてフライトを検索
ArrivalCityDepartureCity DepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0) ｜ (FirstClassSeatsAvailable>0))”	到着都市、出発都市、および出発日に基づいてフライトを検索

ArrivalCityDepartureCity DepartureDateSeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、出発都市、出発日、および席のタイプに基づいてフライトを検索
ArrivalCityDepartureCity SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、出発都市、および席のタイプに基づいてフライトを検索

ArrivalCityDepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着都市と出発日に基づいてフライトを検索
ArrivalCityDepartureDate SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市、出発日、および席のタイプに基づいてフライトを検索
ArrivalCitySeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着都市と席のタイプに基づいてフライトを検索

ArrivalDate()	“SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着日に基づいてフライトを検索
ArrivalDateDepartureCity()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着日と出発都市に基づいてフライトを検索
ArrivalDateDepartureCity DepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着日、出発都市、および出発日に基づいてフライトを検索

ArrivalDateDepartureCity DepartureDateSeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着日、出発都市、出発日、および席のタイプに基づいてフライトを検索
ArrivalDateDepartureCity SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着日、出発都市、および席のタイプに基づいてフライトを検索

ArrivalDateDepartureDate()	“SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	到着日と出発日に基づいてフライトを検索
ArrivalDateDepartureDate SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥””+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥””+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着日、出発日、および席のタイプに基づいてフライトを検索
ArrivalDateSeatType()	• エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	到着日と席のタイプに基づいてフライトを検索

DepartureCity()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum +”¥“) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	出発都市に基づいてフライトを検索
DepartureCityDepartureDate()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	出発都市と出発日に基づいてフライトを検索
DepartureCityDepartureDate SeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	出発都市、出発日、および席のタイプに基づいてフライトを検索

DepartureCitySeatType()	“SELECT (Form = ¥”List Of Available Flights¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	出発都市と席のタイプに基づいてフライトを検索
DepartureDate()	“SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (Status = ¥”New¥“) & ((CoachSeatsAvailable > 0)	(FirstClassSeatsAvailable>0))”	出発日に基づいてフライトを検索
DepartureDateSeatType()	• エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	出発日と席のタイプに基づいてフライトを検索
PrimaryKey()	“SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightByDate = ¥”“+ tempScheduledFlightByDate +”¥“)” “SELECT ((Form = ¥”List Of Available Flights¥“)) & (AvailableFlightNumber = ¥”“ + flightNum + ”¥“)”	スケジュールされた日付別フライトに基づいてフライトを検索

SearchCriteria()	“SELECT (Form = ¥”List Of Available Flights¥“) & (ArrivalCityCode = ¥”“ + aArrivalCity + ”¥“) & (DepartureCityCode = ¥”“ + aDepartureCity + ”¥“)” • エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (ScheduledFlightNumber = ¥”“+ flightNum + ”¥“) & (ScheduledArrivalDate = [” + aArrivalDate + “]) & (ScheduledDepartureDate = [” + aDepartureDate + “]) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	すべての検索条件に基づいてフライトを検索
SeatType()	• エコノミー・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (CoachSeatsAvailable > 0) & (Status = ¥”New¥“)” • ファースト・クラス “SELECT (Form = ¥”Scheduled Flights¥“) & (FirstClassSeatsAvailable > 0) & (Status = ¥”New¥“)”	席のタイプに基づいてフライトを検索

ejbLoad() メソッドは、スケジュールされた日付別フライトに基づいて、「スケジュールされたフライト」文書および対応する「利用可能なフライトのリスト」文書からデータを取得して Bean プロパティーに入れるために使用されます。

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 5 に示します。

表 5 Flight の getter および setter メソッド

getter メソッド	setter メソッド	フォームのフィールド名	データ型
getArrivalCityCode()	setArrivalCityCode()	List Of Available Flights: DisplayOnlyArrival	String

getArrivalDate()	setArrivalDate()	Scheduled Flights: ScheduledArrivalDate	String
getArrivalTime()	setArrivalTime()	List Of Available Flights: ArrivalTime	String
getCoachPrice()	setCoachPrice()	List Of Available Flights: CoachPrice	String
getCoachSeatsAvailable()	setCoachSeatsAvailable()	Scheduled Flights: CoachSeatsAvailable	String
getData()		Enterprise Bean 内の値を収容した FlightInfo JavaBean	FlightInfo
getDepartureCityCode()	setDepartureCityCode()	List Of Available Flights: DisplayOnlyDeparture	String
getDepartureDate()	setDepartureDate()	Scheduled Flights: ScheduledDepartureDate	String
getDepartureTime()	setDepartureTime()	List Of Available Flights: DepartureTime	String
getFirstClassPrice()	setFirstClassPrice()	List Of Available Flights: FirstClassPrice	String
getFirstClassSeatsAvailable()	setFirstClassSeatsAvailable()	Scheduled Flights: FirstClassSeatsAvailable	String
getFood()	setFood()	List Of Available Flights: Food	String
getListAirplaneType()	setListAirplaneType()	List Of Available Flights: ListAirplaneType	String
getScheduledFlightByDate()	setScheduledFlightByDate()	Scheduled Flights: ScheduledFlightByDate	String

FlightInfo

FlightInfo は、フライト情報を保管するために使用される JavaBean です。この JavaBean は適切な JSP に渡されます。JSP はこの JavaBean を使用して、特定のフライト・データを取得します。

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 6 に示します。

表 6 FlightInfo の getter および setter メソッド

getter メソッド	setter メソッド	値	データ型
getArrivalCityCode()	setArrivalCityCode()	フライトの到着都市コード	String
getArrivalDate()	setArrivalDate()	フライトの到着日	String
getArrivalTime()	setArrivalTime()	フライトの到着時刻	String
getCityCode()	setCityCode()	フライトの都市コード	String
getCoachPrice()	setCoachPrice()	フライトのエコノミー・クラスの料金	String
getCoachSeatsAvailable()	setCoachSeatsAvailable()	フライトのエコノミー・クラスの空席数	String

getDepartureCityCode()	setDepartureCityCode()	フライトの出発都市コード	String
getDepartureDate()	setDepartureDate()	フライトの出発日	String
getDepartureTime()	setDepartureTime()	フライトの出発時刻	String
getFirstClassPrice()	setFirstClassPrice()	フライトのファースト・クラスの料金	String
getFirstClassSeatsAvailable()	setFirstClassSeatsAvailable()	フライトのファースト・クラスの空席数	String
getFlightList()	setFlightList()	フライトのリスト	Vector
getFood()	setFood()	フライトの食事	String
getListAirplaneType()	setListAirplaneType()	フライトの飛行機の機種	String
getScheduledFlightByDate()	setScheduledFlightByDate()	フライトのスケジュールされた日付別フライト	String

Ticket Bean

Ticket Enterprise Bean は、チケットを表すために使用される Entity Bean です。この Bean は、チケット・データに対して実行される、次のようなタスクを実行するためのメソッドを提供します。

- 顧客が予約したフライトの新規チケットの作成
- 指定された顧客番号または旅程番号を持つチケットのリストの取得
- 決済済みのチケットに対するマーキング、およびクレジット・カード情報の保管

Ticket Enterprise Bean は、Bean 管理パーシスタンスを使用し、 flights.nsf Lotus Domino データベース内の「チケット情報」フォームおよび「クレジット情報」フォームにマップされます。「チケット情報」フォームのレイアウトは 6 ページの『Lotus(R) Domino(TM) 環境のワークフロー』セクションの表 5 に示されており、「クレジット情報」フォームのレイアウトは、同セクションの表 6 に示されています。

Ticket Enterprise Bean には、複数の ejbFindBy メソッドが含まれています。これらの finder メソッドは、ホーム・インターフェースで定義されています。表 7 に、各 finder メソッドを示します。

表 7 Ticket の Finder メソッド

メソッド名	Select 文	説明
ejbFindByCustomerNumber()	“SELECT ((Form=¥”Ticket Information¥“)) & (TicketCustomerNumber = ¥”“ + aCustomerNumber + ”¥“)”	顧客番号に基づいてチケットを検索
ejbFindByInvoiceNumber()	“SELECT ((Form=¥”Ticket Information¥“)) & (InvoiceNumber = ¥”“ + aInvoiceNumber + ”¥“)”	請求書番号に基づいてチケットを検索

ejbFindByPrimaryKey()	“SELECT ((Form=¥”Ticket Information¥“)) & (FlightByDate = ¥”“ + tk.flightByDate + ”¥“) & (SeatNumber = ¥”“ + tk.seatNumber +”¥“)” “SELECT ((Form=¥”Credit Information¥“)) & (InvoiceNumber = ¥”“ + invoiceNumber + ”¥“)”	日付別フライトおよび座席番号に基づいてチケットを検索

ejbLoad() メソッドは、日付別フライトおよび座席番号に基づき、「チケット情報」文書および対応する「クレジット情報」文書からデータを取得して Bean プロパティーに入れるために使用されます。

ejbStore() メソッドは、Bean プロパティーから、特定の「チケット情報」文書および対応する「クレジット情報」文書に含まれるデータを更新するために使用されます。

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 8 に示します。

表 8 Ticket の getter および setter メソッド

getter メソッド	setter メソッド	フォームのフィールド名	データ型
getCreditCardExpirationDate()	setCreditCardExpirationDate()	Credit Information: ExpirationDate	String
getCreditCardNumber()	setCreditCardNumber()	Credit Information: CreditCardNumber	String
getCreditCardType()	setCreditCardType()	Credit Information: CardType	String
getCustomerNumber()	setCustomerNumber()	Ticket Information: TicketCustomerNumber	String
getData()		Enterprise Bean 内の値を収容した TicketInfo JavaBean	String
getFlightByDate()	setFlightByDate()	Ticket Information: FlightByDate	String
getInvoiceNumber()	setInvoiceNumber()	Ticket Information: InvoiceNumber	String
getPaidStatus()	setPaidStatus	Ticket Information: PaidStatus	String
getPassengerAddress()	setPassengerAddress()	Ticket Information: PassengerStreet	String
getPassengerCity()	setPassengerCity()	Ticket Information: PassengerCity	String
getPassengerCountry()	setPassengerCountry()	Ticket Information: PassengerCountry	String
getPassengerFirstName()	setPassengerFirstName()	Ticket Information: PassengerFirstName	String
getPassengerLastName()	setPassengerLastName()	Ticket Information: PassengerLastName	String
getPassengerMiddleInitial()	setPassengerMiddleInitial()	Ticket Information: PassengerMiddleInitial	String

getPassengerPhoneNumber()	setPassengerPhoneNumber()	Ticket Information: PassengerPhone	String
getPassengerState()	setPassengerState()	Ticket Information: PassengerState	String
getPassengerZipCode()	setPassengerZipCode()	Ticket Information: PassengerZip	String
getSeatNumber()	setSeatNumber()	Ticket Information: SeatNumber	String
getTicketClass()	setTicketClass()	Ticket Information: TicketClass	String
getTicketPrice()	setTicketPrice()	Ticket Information: TicketPrice	BigDecimal
getTicketStatus()	setTicketStatus()	Ticket Information: TicketStatus	String
payTicket()		Ticket Information: PaidStatus and sets Credit Information	String

TicketInfo

TicketInfo は、チケット情報を保管するために使用される JavaBean です。この JavaBean は適切な JSP に渡されます。JSP はこの JavaBean を使用して、特定のチケット・データを取得します。

getter メソッドおよび setter メソッド、およびそれらのメソッドで戻される値や設定される値を、表 9 に示します。

表 9 TicketInfo の getter および setter メソッド

getter メソッド	setter メソッド	値	データ型
getCreditCardExpirationDate()	setCreditCardExpirationDate()	クレジット・カードの有効期限	String
getCreditCardNumber()	setCreditCardNumber()	クレジット・カード番号	String
getCreditCardType()	setCreditCardType()	クレジット・カードの種類	String
getCustomerNumber()	setCustomerNumber()	顧客番号	String
getFlightByDate()	setFlightByDate()	日付別フライト	String
getInvoiceNumber()	setInvoiceNumber()	請求書番号	String
getPaidStatus()	setPaidStatus()	支払状況	String
getPassengerAddress()	setPassengerAddress()	乗客の住所	String
getPassengerCity()	setPassengerCity()	乗客の市区町村	String
getPassengerCountry()	setPassengerCountry()	乗客の国	String
getPassengerFirstName()	setPassengerFirstName()	乗客のファーストネーム	String
getPassengerLastName()	setPassengerLastName()	乗客のラストネーム	String
getPassengerMiddleInitial()	setPassengerMiddleInitial()	乗客のミドル・ネームのイニシャル	String
getPassengerPhoneNumber()	setPassengerPhoneNumber()	乗客の電話番号	String
getPassengerState()	setPassengerState()	乗客の都道府県	String
getPassengerZipCode()	setPassengerZipCode()	乗客の郵便番号	String
getSeatNumber()	setSeatNumber()	座席番号	String
getTicketClass()	setTicketClass()	チケットのクラス	String
getTicketList()	setTicketList()	チケットのリスト	Vector
getTicketPrice()	setTicketPrice()	チケットの料金	String

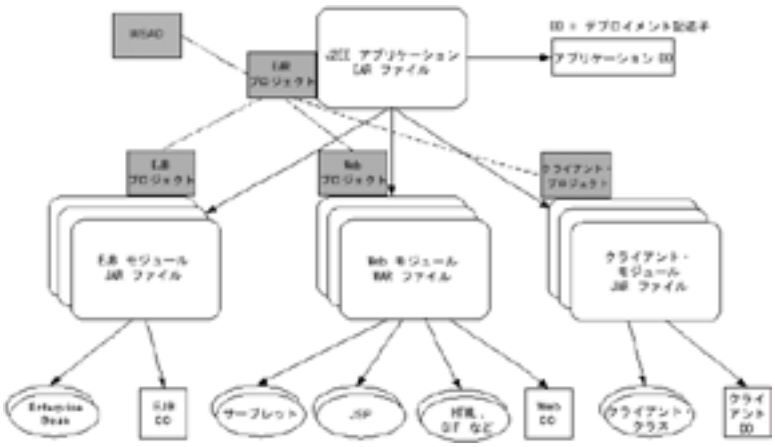
getTicketStatus()	setTicketStatus()	チケットの状況	String
getTotalCost()		チケット・リストのベクター内の全チケットの料金の合計	String
getUniqueInvoiceNumbers()		チケット・リストのベクター内の全チケットの固有の請求書番号	Vector
toString()		オブジェクトのストリング表記	String

エンタープライズ・アプリケーションのインストール

WebSphere Studio Application Developer は、 Visual Age for Java(TM) と WebSphere Studio の両方の機能性を兼ね備えています。このほかに、数多くの新しい機能も追加されています。パースペクティブをサポートしており、アプリケーションをさまざまなビューで扱うことができます。例えば、J2EE パースペクティブでは、 J2EE 対応アプリケーションの構築用にカスタマイズされた環境で作業することができます。

Application Developer を使用すると、ユーザーのアプリケーションを、 Enterprise ARchive ファイル (EAR) や Web アーカイブ・ファイル (WAR) などの J2EE 対応フォーマットで、直接エクスポートすることができます。これらのファイルは、WebSphere Application Assembly Tool (AAT) を使用せずに、WebSphere Application Server 4.0 にエンタープライズ・アプリケーションとしてインストールすることができます。図 2 に、J2EE の階層、およびそれに対応する WebSphere Studio Application Developer のサポートを示します。

図 2 J2EE アーキテクチャー



J2EE アプリケーションは、Enterprise Bean モジュール (Enterprise Bean JAR ファイルに保管される)、Web モジュール (Web アーカイブ (WAR) ファイルに保管される)、およびクライアント・モジュール (JAR ファイルに保管される) を含む、Enterprise ARchive (EAR) ファイルに保管されます。 WAR ファイルには、サーブレット、JSP、HTML ファイル、イメージなどの、 Web アプリケーションのすべてのコンポーネントが含まれます。これらの各モジュールには、デプロイメント記述子が含まれています。例えば、WAR ファイルには、web.xml ファイルが含まれています。

J2EE 階層は、WSAD 内のプロジェクトと対応します。 EAR プロジェクトには、Enterprise Bean プロジェクト、Web プロジェクト、およびクライアント・プロジェクトへの参照が含まれています。

このアプリケーション内には、 FlightsEJBModule、および Web モジュールの FlightsWebModule を含む、FlightsEAR という EAR ファイルがあります。 FlightsEJBModule には、すべての Enterprise Bean (CustomerFlight、Customer、Flight、および Ticket) が含まれています。 FlightWebModule には、すべての Web コンポーネント (FlightsServlet、HTML、JSP、およびイメージ・ファイル) が含まれています。

flights エンタープライズ・アプリケーション・ファイルを生成するには、以下の手順を実行します。

1. Application Developer のメインスクリーンから、「ファイル (File)」->「エクスポート (Export)」を選択する。
2. 「エクスポート (Export)」ウィンドウが表示されたら、「EAR ファイル (EAR file)」を選択する。
3. 次のウィンドウで、「エクスポートするリソースを選択してください (What resource do you want to export?)」フィールドのドロップダウン・リストから、FlightsEAR リソースを選択する。
4. 「リソースをエクスポートする場所を選択してください (Where do you want to export resources to?)」フィールドに、ロケーション SystemName:¥QIBM¥UserData¥WebASAdv4¥instanceName¥installableApps¥flight.ear を入力する。「完了 (Finish)」をクリックする。
5. flights エンタープライズ・アプリケーションがエクスポートされると、フォルダー内にそのファイルが表示されます。

flights エンタープライズ・アプリケーションをインストールするには、次の手順を実行します。

1. コマンド・プロンプト・ウィンドウを開き、管理コンソールを開始する。「コンソール作動可能 (Console Ready)」というメッセージが表示されるまで待機します。
2. コンソールで、ウィザードのアイコンを選択し、「エンタープライズ・アプリケーションのインストール (Install Enterprise Application)」をクリックする。「アプリケーション・モジュールの指定 (Specifying the Application Module)」ウィンドウが表示されます。「アプリケーションのインストール (Install Application)」ラジオ・ボタンが選択されていることを確認します。「パス (Path)」の横にある「参照 (Browse)」ボタンをクリックし、 ¥QIBM¥UserData¥WebASAdv4¥instanceName¥installableApps ディレクトリー内の flight.ear を指定します。
3. 「次へ (Next)」をクリックする。「このアプリケーションには、メソッド許可が含まれています。保護されていないメソッドをすべて拒否しますか? (This application contains method permissions. Do you wish to deny all unprotected methods?)」というメッセージが表示されます。「はい」を選択します。
4. 「user 役割のマッピング (Mapping User Roles)」ウィンドウで「選択 (Select)」ボタンを押し、「全認証済みユーザー (All Authenticated Users)」にチェックマークを付けて「OK」を押す。
5. 「アプリケーション・サーバーの選択 (Selecting Application Servers)」ウィンドウが表示されるまで、「次へ (Next)」をクリックする。このウィンドウで、「モジュール (Module)」ボックス内のすべてのモジュールを選択し、「サーバーの選択 (Select Server)」ボタンを押します。デフォルトのサーバーを選択して「OK」を押します。
6. 「次へ (Next)」をクリックし、「完了 (Finish)」をクリックしてアプリケーションをインストールする。「アプリケーションの再生成 (Regenerate the application)」ダイアログが表示されたら、「いいえ」をクリックします。

flights エンタープライズ・アプリケーションが WebSphere Application Server バージョン 4.0 Advanced Edition にインストールされたので、サーバーを停止する必要があります。サーバーを再始動する前に、サーバーの再始動時に NCSOW.jar がクラスパスで検出されるように、 NCSOW.jar を QIBM/UserData/WebASAdv4/InstanceName/lib/ext ディレクトリーにコピーする必要があります。サーバーを再始動して、新規アプリケーションを作動可能にします。

これで、flights エンタープライズ・アプリケーションが使用可能となります。アプリケーションを使用するには、ブラウザーを開き、URL「http://systemName:port/webapp/Flights/index.html」を入力します。

WebSphere(R) Application Server 環境の主要な発見事項

flights シナリオの WebSphere Application Server 環境を作成および使用した際の主要な発見事項について、以下にリストします。

- ステートフル Session Bean の「セッション・タイムアウト (Session Timeout)」値により大きな値を設定するには、WebSphere Application Server の Application Assembly Tool (AAT) で、以下のステップを実行します。
  1. AAT で、Enterprise Bean JAR または EAR をオープンする。
  2. Session Enterprise Bean までドリルダウンして、これをクリックする。
  3. 右側のペインにある「IBM(R) 拡張 (IBM(R) Extensions)」タブをクリックする。
  4. 「タイムアウト (Timeout)」プロパティーを設定する。
  5. JAR または EAR を保管する。
  6. AAT を終了する。
- WebSphere Application Server の Application Assembly Tool (AAT) を使用して、エンタープライズ・アプリケーションのコードを生成することができます。インストール時にエンタープライズ・アプリケーションのコード生成で問題が発生した場合、WebSphere Application Server 管理コンソールでは、インストールが失敗した原因に関する詳しい情報は提供されません。このような場合には、AAT を使用して、同じ EAR ファイルのデプロイメント・コードを生成することができます。また、失敗の原因に関する詳しい情報も提供されます。 WebSphere Application Server 管理コンソールと AAT はいずれも、同じ基礎コードを使用して、コードを生成します。また、「検査 (Verify)」オプションも用意されており、ユーザーは EAR ファイルに対してこのオプションを実行し、追加情報を得ることができます。

- JSP のニーズに対応するために、一部の Lotus(R) Domino(TM) 文書で、フィールドを文書に追加する必要があります。 flightSearch.jsp では、「スケジュールされたフライト」フォームに、 Departure Date フィールドおよび Arrival Date フィールドを追加する必要がありました。 passengerInformation.jsp では、「チケット情報」フォームに、顧客情報関連のフィールドを追加する必要がありました。

- Lotus Domino JDBC ドライバーを使用する場合、 SQL ステートメント内では、Lotus Domino 文書の名前および列の名前において大文字小文字が区別されます。

- flights アプリケーションでは、Lotus Domino Java(TM) API を使用して、Lotus Domino データベースに接続しました。当初は、Lotus Domino JDBC ドライバーを使用する予定でしたが、 WebSphere Application Server では、iSeries(TM) 上で Lotus Domino JDBC ドライバーを使用するデータ・ソースを

作成できません。また、 iSeries では Lotus Domino JDBC ドライバーがまだ移植されていないため、独自の接続プールをインプリメントすることもできませんでした。 iSeries 対応の Lotus Domino JDBC ドライバーは、将来のリリースで提供される予定です。

- 以下のヒントは、WebSphere Java アプリケーションからリモートの Lotus Domino サーバーに接続する際に問題が発生した場合に役立ちます。

  – DIIOP が Lotus Domino サーバーにセットアップされていることを確認してください。 DIIOP がセットアップされていないと、リモート・ホストが接続を拒否したことを示すメッセージが表示されます。

  – NotesFactory セッションの取得を試行する際には、必ず Lotus Domino HTTP サーバーのポート番号を指定してください。例:
  Session session = NotesFactory.createSession("systemName.domainName:portNumber", "user ID", "password").
  これにより、CORBA 要求が、正しいサーバーに送信されます。ポート番号が指定されていないと、NotesException が返されます。

  – 次のような NotesException 4377 例外が返される場合があります : 以下のコード行を使用して Java アプリケーションから getDatabase ステートメントを実行するときには、サーバーがセッションと同じホストでなければなりません (Server must be on same host as session when performing the getDatabase statement from a Java Application using the following lines of code) :
  Session session = NotesFactory.createSession(notesServer, notesUser, password);
  ndbContent = session.getDatabase(notesServer, notesDatabase);

    これには、以下の 2 つの解決方法があります。

    1. getDatabase で、ポート番号と共にサーバー名を送るのではなく、サーバー名のみを送信する。
    2. getDatabase 呼び出しで、以下のように notesServer として "" を渡す。これにより、作成されたセッションの使用が強制されます。
       ndbContent = session.getDatabase("", notesDatabase);

  – 使用している Lotus Domino のバージョンから取得した NCSOW.jar ファイルが、CLASSPATH に含まれていることを確認します。これは、NCSO.jar ファイルの WebSphere バージョンです。 NCSO.jar は使用に適していません。これは IIOP レベルが競合して、セッションを作成できないためです。

  – Domino Java API を使用して IIOP 経由で WebSphere Java アプリケーションから Domino データベースに接続するには、アプリケーションが lotus.domino.* パッケージをインポートする必要があります。

  – NotesException に関する追加情報を得るには、以下のコードの System.out.println を追加してください。これにより、エラーを詳細に説明する静的変数を表示させることができます。

```
catch(lotus.domino.NotesException  ne)
{
System.out.println(ne.text  +  “  ”  +  ne.id);
ne.printStackTrace();
}
```

- Lotus Domino Java API を使用する Java アプリケーションで、日付に対する検索を実行するための Select ステートメントを記述する場合は、検索する日付を定数として指定する必要があります (つまり、値を [ ] で囲みます)。例: SELECT (Form = “Scheduled Flights”) & (ScheduledDepartureDate = [10/31/2001])

- Bean 管理パーシスタンス Entity Bean 内で finder メソッドをコーディングすると、その finder は、ejbFindBy*xxxx*() メソッドによってインプリメントされます。ホーム・インターフェース内の finder メソッドは、findBy*xxxx*() という名前になります。このコンテナーは、 findBy*xxxx*() を、実質的には ejbFindBy*xxxx*() を囲むラッパーとして実装するため、実際の finder のコードを記述する必要がありません。

- Bean 管理パーシスタンス Entity Bean を Session Bean から呼び出すときに、CSITransactionRolledbackException を受信しないようにするには、Session Bean を、トランザクション属性に TX_SUPPORTS を設定してデプロイする必要があります。

  org.omg.CORBA.INV_OBJREF 例外は、クライアント・コードが、Lotus Domino サーバーの認識していないオブジェクトを使用しようとしている場合に発生します。これは、クライアント・セッションが、サーバー・レコード内のセッション・タイムアウト・パラメーターで許可されている時間よりも長くアイドル状態のままになっている場合か、誰かがサーバー・コンソールからすべてのセッションを強制的にドロップしようとした場合に発生する可能性があります。セッションを長時間アイドルのままにしておくことは望ましくないため、 DIIOP が送られてこれらのセッションを終了させます。クライアントが、この状態の処理を行います。

  クライアントがこのエラーを処理しない場合、Lotus Domino Session はタイムアウトし、セッション・オブジェクトは無効になります。以下のエラー・メッセージが表示されます。

  WebSphere Application Server からのエラー・メッセージを、以下に示します。

  ```
  org.omg.CORBA.INV_OBJREF: minor code: 1229062208 completed: No
  java/lang/Throwable.<init>(Ljava/lang/String;)V+4  (Throwable.java:94)
  org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;ILorg/omg/CORBA
  /CompletionStatus;)V+1
  (INV_OBJREF.java:72)
  org/omg/CORBA/INV_OBJREF.<init>(Ljava/lang/String;)V+6  (INV_OBJREF.java:48)
  com/ibm/CORBA/iiop/ReplyMessage.getSystemException()Lorg/omg/CORBA/
  ```

SystemException;+119
(ReplyMessage.java:181)
com/ibm/rmi/iiop/ClientResponseImpl.getSystemException()Lorg/omg/CORBA/
SystemException;+11
(ClientResponseImpl.java:89)
com/ibm/CORBA/iiop/ClientDelegate.invoke(Lorg/omg/CORBA/Object;Lorg/omg/
CORBA/portable/OutputStream;)Lorg/omg/CORBA/portable/InputStream;+235
(ClientDelegate.java:439)
org/omg/CORBA/portable/ObjectImpl._invoke(Lorg/omg/CORBA/portable/
OutputStream;)Lorg/omg/CORBA/portable/InputStream;+4
(ObjectImpl.java:251)
lotus/domino/corba/_IDatabaseStub.search(Ljava/lang/String;Llotus/domino/
corba/IDateTime;I)Llotus/domino/corba/DCData;+0
(_IDatabaseStub.java:0)
lotus/domino/cso/Database.search(Ljava/lang/String;Llotus/domino/DateTime;I)
Llotus/domino/DocumentCollection;+0
(Database.java:1478)
lotus/domino/cso/Database.search(Ljava/lang/String;)Llotus/domino/
DocumentCollection;+0
(Database.java:1452)
com/flights/ejb/session/CustomerFlightBean.getAllCityCodes()Lcom/flights/
FlightInfo;+0
(CustomerFlightBean.java:110)
com/flights/ejb/session/EJSRemoteCustomerFlight.getAllCityCodes()Lcom/
flights/FlightInfo;+0
(EJSRemoteCustomerFlight.java:31)
com/flights/ejb/session/_EJSRemoteCustomerFlight_Tie._invoke(Ljava/lang/
String;Lorg/omg/CORBA/portable/InputStream;Lorg/omg/CORBA/portable/
ResponseHandler;)Lorg/omg/CORBA/portable/OutputStream;+0
(_EJSRemoteCustomerFlight_Tie.java:82)
com/ibm/CORBA/iiop/ExtendedServerDelegate.dispatch(Lcom/ibm/rmi/
ServerRequest;)Lcom/ibm/rmi/ServerResponse;+224
(ExtendedServerDelegate.java:506)
com/ibm/CORBA/iiop/ORB.process(Lcom/ibm/rmi/ServerRequest;)Lcom/ibm/
rmi/ServerResponse;+20
(ORB.java:2282)
com/ibm/CORBA/iiop/WorkerThread.run()V+89 (WorkerThread.java:195)
com/ibm/ejs/oa/pool/ThreadPool$PooledThread.run()V+67 (ThreadPool.java:641)

Domino サーバーからのエラー・メッセージを、以下に示します。

DIIOP SYSTEM EXCEPTION: INV_OBJREF, minor
code 49420040, SOMDERROR_BadObjref [somd_refdata_to_obj
(CORBA::ReferenceData*):1091]

IIOP のタイムアウトをより大きな値に設定したり、エラーをキャッチしてセッションを再度確立してみることができます。

Java クライアントから Lotus Domino セッション・オブジェクトが開いているかどうかを判別するために用いようとした方法について、以下に示します。

– Enterprise Bean 内で例外をキャッチして、そこでそれを処理する。この方法は、以下の理由から機能しません。

  Enterprise JavaBean 1.1 の仕様、セクション 12.3.4 『Exceptions and transactions』からの引用です。

  “クライアントのトランザクション・コンテキストでインスタンスを実行中に、チェックなし例外をスローした場合、コンテナーは、ロールバックのためにそのトランザクションにマークを付け、クライアントに対して、javax.jts.TransactionRolledbackException をスローします。”

  “コンテナーが、何らかの理由でロールバックのためにトランザクションにマークを付けることを決定した場合は、クライアントに対して、javax.jts.TransactionRolledbackException をスローしなければなりません。 javax.jts.TransactionRolledbackException は、 java.rmi.RemoteException のサブクラスです。この例外は、クライアントに対して、トランザクションをコミットできないために、トランザクション内で試行された例外のリカバリーは効果がない、ということを通知します。”

  つまり、Enterprise Bean 内でチェックなし例外が発生すると、明示的にその例外を処理している場合でも、常に不完了トランザクションがロールバックされます。基本的に、チェックなし例外とは、java.lang.Exception から派生するものではないすべての例外です。 CORBA.INV_OBJREF 例外は、java.lang.Exception からは派生しません。このため、チェックなし例外として分類される可能性があります。したがって、コンテナーは、この例外がスローされると、常に TransactionRolledbackException をスローします。

– Lotus Domino Java API の将来のバージョンに追加される Session.isValid() を使用する。これは、セッションの状態を判別するうえでの最良の方法ですが、現時点では使用できません。

  機能: Session.isValid()
  これは、DIIOP 接続プール機能の一部です。

  目的:
  Java 専用のこのメソッドの目的は、作成されたセッション・オブジェクトがまだ有効であるかどうかを判別することです。リモート API では、このメソッドは DIIOP サーバー・タスクがこのセッションを有効とみなしているかどうかを判別します。したがって、このメソッドはネットワーク操作を実行する可能性があります。このような理由から、このメソッドは短いループ内では使用すべきではあ

りません。
ローカル API の場合も、このメソッドは、セッションがまだ有効であるかどうかを判別します。

シグニチャー:
boolean isValid();

注: このメソッドは、いかなる Exception もスローしません。

使用法:

以下は、ワーカー・タイプのスレッドのサーブレット内でのこのメソッドの使用例です。

```
class WorkerThread extends Thread {
...
public void run()
{
while (WaitForWork()) {
if ( ! session.isValid() ) {
// need to create new session
}
// do the work
}
}
}
```

  - Lotus Domino への新規セッションを常に取得するよう、getConnection のコードを修正する。これにより、CORBA.INV_OBJREF 例外はすべて回避されます。 Java アプリケーションに対して追加のオーバーヘッドをもたらしますが、この方法は有効です。

- 「IIOP」タブにあるサーバー文書で、管理者が指定できるスレッドの最大数には制限がありません。Lotus Domino の今後のリリースでは、DIIOP はこの設定を使用しません。これは、下位互換性を保つ目的でのみ、サーバー文書に残されます。 Lotus Domino サーバー文書では、セッションの最小タイムアウト値は 5 分です。

- AAT を使用せずに、QIBM にあるオブジェクトについてのクラスパス情報を指定するには、WebSphere Studio Application Developer で、以下のロケーションに MANIFEST.MF ファイルを作成します。

  Enterprise Bean モジュールでは、次のロケーションで、マニフェスト・ファイルを作成するか、既存のマニフェスト・ファイルを使用してください: ejbModule -> META-INF -> MANIFEST.MF

  Web モジュールでは、次のロケーションで、マニフェスト・ファイルを作成するか、既存のマニフェスト・ファイルを使用してください: webApplication -> META-INF -> MANIFEST.MF

Manifest.mf ファイル内でのクラスパスの表示例を、以下に示します。

```
Manifest-Version: 1.0
Class-Path: /qibm/userdata/webasadv4/flight4/installedapps/flightsear.ear/flightsejbmodule.jar
```

- flights JSP のパブリッシュ済みバージョンを WebSphere Studio Application Developer へインポートした後、各 JSP は Page Designer を使用して編集されました。編集後、EAR がインストール可能なディレクトリーにエクスポートされてから、各 JSP が中央チームのロケーションにエクスポートされました。更新が加えられた JSP は、チームのロケーションにエクスポートされませんでした。また、これらの JSP は、WebSphere Studio Application Developer 内でコピーおよび削除できないことも分かりました。その際、リソースがファイル・システムと同期していないというエラーが返されました。開かれても変更されなかった JSP については、問題ありませんでした。この Web モジュールは、その後ローカル側から更新されました。これにより、JSP をコピー、削除、およびエクスポートできるようになりました。

- 表 1 に、WebSphere Studio Application Developer の Java エディターで使用できるキーボード・ショートカットのリストを示します。

表 1 キーボード・ショートカット

説明	キー・シーケンス
インポート	Ctrl+Shift+M
行番号へジャンプ	Ctrl+L
強調表示されたテキストのインデント	Ctrl+I
検索/置換	Ctrl+F
コピー	Ctrl+C
切り取り	Ctrl+X
元に戻す	Ctrl+Z
すべて選択	Ctrl+A
次のエラーへジャンプ	Ctrl+E
検索テーブルで選択されている項目で、Java 検索を起動する	Ctrl+H
コーディング/コンテンツ・アシスタントを起動する (選択後、Javadoc が吹き出しヘルプ内に表示されます。)	Ctrl+Space
ナビゲーション・ビューで、プロジェクトの増分ビルドを実行する	Ctrl+B
Ctrl キーを押したままリソースをドラッグ・アンド・ドロップして、異なるワークベンチ・ウィンドウ間でリソースをコピーする	Ctrl+ ドラッグ・アンド・ドロップ

第 5 章 参照

- 「WebSphere J2EE Application Development for the IBM eServer iSeries Server」IBM Redbook (SG24-6559-00)
- 「WebSphere Studio Application Developer Programming Guide」IBM Redbook (SG24-6585-00)
- 「Tips for Working with Domino Objects」 http://www.advisor.com/Articles.nsf/aidp/BALAB03
- 「WebSphere Studio Application Developer Migration Guides」 http://www7b.boulder.ibm.com/wsdd/library/techarticles/0110_wsad_mig/migration_ga.html

例 : Customer Bean

以下の例では、Customer Entity Bean のコードを示します。 Customer Bean は、Bean 管理パーシスタンスを使用し、 names.nsf Lotus(R) Domino(TM) データベース内の FlightPerson フォームにマップされます。 Customer Bean は、Lotus Domino API を使用して、Lotus Domino データベースにアクセスします。

CustomerKey のソース・コードを、例 1 に示します。

例 1: CustomerKey のソース・コード

```
package com.flights.ejb.bmp;


/**
* This is a Primary Key Class for the Entity Bean
**/
public class CustomerKey implements java.io.Serializable {
public String primaryKey;
final static long serialVersionUID = 3206093459760846163L;


/**
* CustomerKey() constructor
*/
public CustomerKey() {
}
/**
* CustomerKey(String key) constructor
*/
public CustomerKey(String key) {
primaryKey = key;
}
```

```
/**
* equals method
* - user must provide a proper implementation for the equal method. The generated
* method assumes the key is a String object.
*/
public boolean equals (Object o) {
if (o instanceof CustomerKey)
return primaryKey.equals(((CustomerKey)o).primaryKey);
else
return false;
}
/**
* hashode method
* - user must provide a proper implementation for the hashCode method. The generated
* method assumes the key is a String object.
*/
public int hashCode () {
return primaryKey.hashCode();
}
}
```

CustomerHome インターフェースのソース・コードを、例 2 に示します。

例 2: CustomerHome のソース・コード

```
package com.flights.ejb.bmp;

/**
* This is a Home interface for the Entity Bean
*/
public interface CustomerHome extends javax.ejb.EJBHome {

/**
* create method for a BMP entity bean
* @return com.flights.ejb.bmp.Customer
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
com.flights.ejb.bmp.Customer create(com.flights.ejb.bmp.CustomerKey primaryKey) throws
javax.ejb.CreateException, java.rmi.RemoteException;
```

```
/**
* findByPrimaryKey method comment
* @return com.flights.ejb.bmp.Customer
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
com.flights.ejb.bmp.Customer findByPrimaryKey(com.flights.ejb.bmp.CustomerKey key) throws
java.rmi.RemoteException, javax.ejb.FinderException;
}
```

Customer リモート・インターフェースのソース・コードを、例 3 に示します。

例 3: Customer リモート・インターフェースのソース・コード

```
package com.flights.ejb.bmp;


/**
* This is the enterprise bean Remote Interface for the Customer bean
*/
public interface Customer extends javax.ejb.EJBObject {


/**
* Returns the address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerAddress() throws java.rmi.RemoteException;
/**
* Returns the city for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCity() throws java.rmi.RemoteException;
/**
* Returns the country for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerCountry() throws java.rmi.RemoteException;
/**
```

```
* Returns the first name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerFirstName() throws java.rmi.RemoteException;
/**
* Returns the internet address for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerInternetAddress() throws java.rmi.RemoteException;
/**
* Returns the last name of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerLastName() throws java.rmi.RemoteException;
/**
* Returns the middle initial of the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerMiddleInitial() throws java.rmi.RemoteException;
/**
* Returns the customer number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerNumber() throws java.rmi.RemoteException;
/**
* Returns the customer phone number.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerPhoneNumber() throws java.rmi.RemoteException;
/**
* Returns the state for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerState() throws java.rmi.RemoteException;
/**
* Returns the zip code for the customer.
* @return java.lang.String
* @exception String The exception description.
*/
java.lang.String getCustomerZipCode() throws java.rmi.RemoteException;
/**
```

```
* Returns the values within the bean as data stored within a CustomerInfo JavaBean.
* @return com.flights.CustomerInfo
* @exception String The exception description.
*/
com.flights.CustomerInfo getData() throws java.rmi.RemoteException;
/**
* Sets the address for the customer.
* @return void
* @param newCustomerAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerAddress(java.lang.String newCustomerAddress) throws java.rmi.RemoteException;
/**
* Sets the city for the customer.
* @return void
* @param newCustomerCity java.lang.String
* @exception String The exception description.
*/
void setCustomerCity(java.lang.String newCustomerCity) throws java.rmi.RemoteException;
/**
* Sets the country for the customer.
* @return void
* @param newCustomerCountry java.lang.String
* @exception String The exception description.
*/
void setCustomerCountry(java.lang.String newCustomerCountry) throws java.rmi.RemoteException;
/**
* Sets the first name of the customer.
* @return void
* @param newCustomerFirstName java.lang.String
* @exception String The exception description.
*/
void setCustomerFirstName(java.lang.String newCustomerFirstName) throws java.rmi.RemoteException;
/**
* Sets the internet address for the customer.
* @return void
* @param newCustomerInternetAddress java.lang.String
* @exception String The exception description.
*/
void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) throws java.rmi.RemoteException;
/**
* Sets the last name of the customer.
* @return void
* @param newCustomerLastName java.lang.String
* @exception String The exception description.
*/
void setCustomerLastName(java.lang.String newCustomerLastName) throws java.rmi.RemoteException;
/**
```

```
* Sets the middle initial of the customer.
* @return void
* @param newCustomerMiddleInitial java.lang.String
* @exception String The exception description.
*/
void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) throws java.rmi.RemoteException;
/**
* Sets the customer number.
* @return void
* @param newCustomerNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerNumber(java.lang.String newCustomerNumber) throws java.rmi.RemoteException;
/**
* Sets the phone number for the customer.
* @return void
* @param newCustomerPhoneNumber java.lang.String
* @exception String The exception description.
*/
void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) throws java.rmi.RemoteException;
/**
* Sets the state for the customer.
* @return void
* @param newCustomerState java.lang.String
* @exception String The exception description.
*/
void setCustomerState(java.lang.String newCustomerState) throws java.rmi.RemoteException;
/**
* Sets the zip code for the customer.
* @return void
* @param newCustomerZipCode java.lang.String
* @exception String The exception description.
*/
void setCustomerZipCode(java.lang.String newCustomerZipCode) throws java.rmi.RemoteException;
}
```

Customer Enterprise Bean のソース・コードを、例 4 に示します。

例 4: Customer Enterprise Bean のソース・コード

```
package com.flights.ejb.bmp;
```

```
import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;
import lotus.domino.*;
import javax.naming.*;
/**
* This is an Entity Bean class with BMP fields
*/
public class CustomerBean implements EntityBean {
private javax.ejb.EntityContext entityContext = null;
private final static long serialVersionUID = 3206093459760846163L;



private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerCountry;
private java.lang.String customerFirstName;
private java.lang.String customerInternetAddress;
private java.lang.String customerLastName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerPhoneNumber;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerNumber;
private transient Database ndbContent = null;
/**
* ejbActivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbActivate() throws java.rmi.RemoteException {}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException
{
return null;
}
/**
* ejbCreate method for a BMP entity bean
* @return com.flights.ejb.bmp.CustomerKey
* @param key com.flights.ejb.bmp.CustomerKey
* @exception javax.ejb.CreateException The exception description.
* @exception java.rmi.RemoteException The exception description.
```

```
*/
public com.flights.ejb.bmp.CustomerKey ejbCreate(com.flights.ejb.bmp.CustomerKey key) throws
javax.ejb.CreateException, java.rmi.RemoteException {



return null;
}
/**
* ejbFindByPrimaryKey method comment
* @return com.flights.ejb.bmp.CustomerKey
* @param primaryKey com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
public com.flights.ejb.bmp.CustomerKey ejbFindByPrimaryKey(com.flights.ejb.bmp.CustomerKey primaryKey)
throws java.rmi.RemoteException, javax.ejb.FinderException {
refresh(primaryKey);
return primaryKey;
}
/**
* Used to refresh the enterprise bean from the persistent storage.
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbLoad() throws java.rmi.RemoteException {



System.out.println("Customer.ejbLoad()");
try
{
refresh((CustomerKey) entityContext.getPrimaryKey());
}
catch (FinderException fe)
{
throw new RemoteException(fe.getMessage());
}
}
/**
* ejbPassivate method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPassivate() throws java.rmi.RemoteException {}
/**
* ejbPostCreate method for a BMP entity bean
* @param key com.flights.ejb.bmp.CustomerKey
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbPostCreate(com.flights.ejb.bmp.CustomerKey key) throws java.rmi.RemoteException {}
```

```
/**
* ejbRemove method comment — currently not implemented
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.RemoveException The exception description.
*/
public void ejbRemove() throws java.rmi.RemoteException, javax.ejb.RemoveException {}
/**
* ejbStore method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void ejbStore() throws java.rmi.RemoteException {



System.out.println("Customer.ejbStore() ");



DocumentCollection dclResult = null;
Document docResult = null;



try
{
ndbContent = getConnection();



// Search for document with specified key values
System.out.println("Searching for document: " + customerNumber);
dclResult = ndbContent.search("SELECT (Form = ¥"FlightPerson¥") & (PersonalID = "+
customerNumber.trim() +")");
docResult = dclResult.getFirstDocument();



if (docResult != null)
{
// Update document to contain new values
System.out.println("Should be updating customer info");



docResult.replaceItemValue("FirstName", customerFirstName);
docResult.replaceItemValue("MiddleInitial", customerMiddleInitial);
docResult.replaceItemValue("LastName", customerLastName);
docResult.replaceItemValue("StreetAddress", customerAddress);
docResult.replaceItemValue("City", customerCity);
docResult.replaceItemValue("State", customerState);
docResult.replaceItemValue("Zip", customerZipCode);
```

```
docResult.replaceItemValue("Country", customerCountry);
docResult.replaceItemValue("PhoneNumber", customerPhoneNumber);
docResult.replaceItemValue("InternetAddress", customerInternetAddress);



docResult.save();
}
else
throw new FinderException("Customer EJB Store: CustomerBean (" + customerNumber + ") not found");



System.out.println("After update");



}
catch(lotus.domino.NotesException ne)
{
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e)
{
e.printStackTrace();
throw new RemoteException(e.toString());
}



finally
{
try
{
System.out.println("Recycle objects");
if (dclResult != null)
dclResult.recycle();
if (docResult != null)
docResult.recycle();
}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}
```

```
}
/**
* Used to return a connection to the Lotus Domino database using Lotus Domino APIs .
* Creation date: (10/19/2001 3:14:11 PM)
* @return javax.sql.DataSource
* @exception java.sql.SQLException The exception description.
*/
private Database getConnection() throws java.rmi.RemoteException, java.sql.SQLException {


if (ndbContent == null) {
Properties properties = getEntityContext().getEnvironment();
String providerURL = properties.getProperty("provider_url");
String notesServer = properties.getProperty("notesServer");
String notesUser = properties.getProperty("notesUser");
String password = properties.getProperty("password");
String notesDatabase = properties.getProperty("notesDB");


InitialContext ctx = null;
Properties prop = new Properties();


try {
prop.put(Context.PROVIDER_URL, providerURL);
prop.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
ctx = new InitialContext(prop);


System.out.println("creating notes session using current creds");
Session session = NotesFactory.createSession(notesServer, null);
System.out.println("username: " + session.getUserName());


System.out.println("got session - getting DB");
ndbContent = session.getDatabase("", notesDatabase);


System.out.println("got database");


if (!ndbContent.isOpen()) ndbContent.open();
}
catch(lotus.domino.NotesException ne){
System.out.println(ne.text + " " + ne.id);
```

```
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch (Exception e) {
System.out.println("an error occurred");
e.printStackTrace();
throw new RemoteException(e.toString());
}
}



return ndbContent;
}
/**
* Returns address for customer.
* Creation date: (04/02/02 2:37:06 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
* Returns city for customer.
* Creation date: (04/02/02 2:37:28 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (04/02/02 2:37:46 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (04/02/02 2:38:05 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address of customer.
* Creation date: (04/02/02 2:38:24 PM)
```

```
* @return java.lang.String
*/
public java.lang.String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
* Creation date: (04/02/02 2:38:39 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns middle initial of customer.
* Creation date: (04/02/02 2:38:55 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerNumber() {
return customerNumber;
}
/**
* Returns phone number for customer.
* Creation date: (04/02/02 2:39:10 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (04/02/02 2:39:22 PM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (04/02/02 2:39:40 PM)
```

```
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
/**
* Returns the values within the enterprise bean as data stored within a CustomerInfo JavaBean.
* Creation date: (03/26/02 7:20:12 AM)
*/
public com.flights.CustomerInfo getData() {


com.flights.CustomerInfo customerInfo = new com.flights.CustomerInfo();


customerInfo.setCustomerFirstName(customerFirstName);


System.out.println("getData(): Here's the customer first name: " + customerFirstName);


customerInfo.setCustomerMiddleInitial(customerMiddleInitial);


System.out.println("getData(): Here's the customer middle initial: " + customerMiddleInitial);
customerInfo.setCustomerLastName(customerLastName);
customerInfo.setCustomerAddress(customerAddress);
customerInfo.setCustomerInternetAddress(customerInternetAddress);


System.out.println("getData(): Here's the customer internet address: " + customerInternetAddress);
customerInfo.setCustomerCity(customerCity);
customerInfo.setCustomerState(customerState);
customerInfo.setCustomerZipCode(customerZipCode);
customerInfo.setCustomerCountry(customerCountry);
customerInfo.setCustomerPhoneNumber(customerPhoneNumber);


return customerInfo;
}
/**
* getEntityContext method comment
* @return javax.ejb.EntityContext
*/
```

```
public javax.ejb.EntityContext getEntityContext() {
return entityContext;
}


/**
* Refreshes the enterprise bean corresponding to the primary key from the persistent
* storage.
* Creation date: (11/14/2001 2:06:39 PM)
* @param aPrimaryKey com.flights.ejb.bmp.FlightKey
* @exception java.rmi.RemoteException The exception description.
* @exception javax.ejb.FinderException The exception description.
*/
private void refresh(CustomerKey primaryKey) throws java.rmi.RemoteException, javax.ejb.FinderException {


DocumentCollection customerDC = null;
Document customerDoc = null;


System.out.println("starting refresh");


if (primaryKey == null)
throw new RemoteException("Primary key cannot be null");


customerNumber = primaryKey.primaryKey;
System.out.println("Customer Number in refresh: " + customerNumber);


try {


ndbContent = getConnection();


// find the customer number to obtain customer info
customerDC = ndbContent.search("SELECT (Form = ¥"FlightPerson¥") & (PersonalID = "+
customerNumber.trim() +")");
System.out.println("number of elements in collection: " + customerDC.getCount());
customerDoc = customerDC.getFirstDocument();
```

```
if (customerDoc != null) {
customerFirstName = customerDoc.getItemValueString("FirstName");
customerMiddleInitial = customerDoc.getItemValueString("MiddleInitial");
customerLastName = customerDoc.getItemValueString("LastName");
customerAddress = customerDoc.getItemValueString("StreetAddress");
customerCity = customerDoc.getItemValueString("City");
customerState = customerDoc.getItemValueString("State");
customerZipCode = customerDoc.getItemValueString("Zip");
customerCountry = customerDoc.getItemValueString("Country");
customerPhoneNumber = customerDoc.getItemValueString("PhoneNumber");
customerInternetAddress = customerDoc.getItemValueString("InternetAddress");


System.out.println("FN: " + customerFirstName + " LN: " + customerLastName + " MI: " +
customerMiddleInitial + " Internet Addr: " + customerInternetAddress);


} // end if
else
{
throw new FinderException("Customer information not found for customer number " + customerNumber);


}


} // end try
catch(lotus.domino.NotesException ne){
System.out.println(ne.text + " " + ne.id);
ne.printStackTrace();
throw new RemoteException(ne.toString());
}
catch(Exception e) {
e.printStackTrace();
throw new RemoteException(e.toString());
}
finally
{
try
{
System.out.println("Recycle objects in CustomerBean refresh");
if (customerDC != null)
customerDC.recycle();
if (customerDoc != null)
customerDoc.recycle();
```

```
}
catch(Exception ex)
{
throw new RemoteException(ex.toString());
}
}
}
/**
* Sets the address for the customer.
* Creation date: (04/02/02 2:37:06 PM)
* @param newCustomerAddress java.lang.String
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Sets the city for the customer.
* Creation date: (04/02/02 2:37:28 PM)
* @param newCustomerCity java.lang.String
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets the country for the customer.
* Creation date: (04/02/02 2:37:46 PM)
* @param newCustomerCountry java.lang.String
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
customerCountry = newCustomerCountry;
}
/**
* Sets the first name of the customer.
* Creation date: (04/02/02 2:38:05 PM)
* @param newCustomerFirstName java.lang.String
*/
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets the internet address for the customer.
* Creation date: (04/02/02 2:38:24 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(java.lang.String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
```

```
* Sets the last name of the customer.
* Creation date: (04/02/02 2:38:39 PM)
* @param newCustomerLastName java.lang.String
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets the middle initial of the customer.
* Creation date: (04/02/02 2:38:55 PM)
* @param newCustomerMiddleInitial java.lang.String
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets the customer number.
* Creation date: (04/02/02 2:40:12 PM)
* @param newCustomerNumber java.lang.String
*/
public void setCustomerNumber(java.lang.String newCustomerNumber) {
customerNumber = newCustomerNumber;
}
/**
* Sets the phone number for the customer.
* Creation date: (04/02/02 2:39:10 PM)
* @param newCustomerPhoneNumber java.lang.String
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets the state for the customer.
* Creation date: (04/02/02 2:39:22 PM)
* @param newCustomerState java.lang.String
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets the zip code for the customer.
* Creation date: (04/02/02 2:39:40 PM)
* @param newCustomerZipCode java.lang.String
*/
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
/**
```

```
* setEntityContext method comment
* @param ctx javax.ejb.EntityContext
* @exception java.rmi.RemoteException The exception description.
*/
public void setEntityContext(javax.ejb.EntityContext ctx) throws java.rmi.RemoteException {
entityContext = ctx;
}
/**
* unsetEntityContext method comment
* @exception java.rmi.RemoteException The exception description.
*/
public void unsetEntityContext() throws java.rmi.RemoteException {
entityContext = null;
}
}
```

CustomerInfoBean のソース・コードを、例 5 に示します。

例 5: CustomerInfoBean のソース・コード

```
package com.flights;
import java.util.*;

/**
* CustomerInfo is a JavaBean used to store the customer information for
* a specific customer/flight. It is passed to the appropriate
* JSP which will use it to retrieve the specific customer data.
*/
public class CustomerInfo implements java.io.Serializable {

private java.lang.String customerFirstName;
private java.lang.String customerMiddleInitial;
private java.lang.String customerLastName;
private java.lang.String customerAddress;
private java.lang.String customerCity;
private java.lang.String customerState;
private java.lang.String customerZipCode;
private java.lang.String customerCountry;
private java.lang.String customerPhoneNumber;
private java.lang.String customerInternetAddress;
private Vector customerListVector;
```

```
/**
* CustomerInfo constructor.
*/
public CustomerInfo() {
super();
}
/**
* Returns address for customer.
* Creation date: (02/11/02 10:08:49 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerAddress() {
return customerAddress;
}
/**
* Returns city for customer.
* Creation date: (02/11/02 10:10:19 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCity() {
return customerCity;
}
/**
* Returns country for customer.
* Creation date: (02/11/02 10:11:38 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerCountry() {
return customerCountry;
}
/**
* Returns first name of customer.
* Creation date: (02/11/02 10:28:15 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerFirstName() {
return customerFirstName;
}
/**
* Returns internet address for customer.
* Creation date: (04/02/02 12:31:05 PM)
* @return java.lang.String
*/
public String getCustomerInternetAddress() {
return customerInternetAddress;
}
/**
* Returns last name of customer.
```

```
* Creation date: (02/11/02 10:29:22 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerLastName() {
return customerLastName;
}
/**
* Returns list of customers.
* Creation date: (02/11/02 10:30:00 AM)
* @return java.lang.String
*/
public Vector getCustomerListVector() {
return customerListVector;
}
/**
* Returns middlet initial of customer.
* Creation date: (02/11/02 10:30:00 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerMiddleInitial() {
return customerMiddleInitial;
}
/**
* Returns phone number for customer.
* Creation date: (02/11/02 10:30:28 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerPhoneNumber() {
return customerPhoneNumber;
}
/**
* Returns state for customer.
* Creation date: (02/11/02 10:31:01 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerState() {
return customerState;
}
/**
* Returns zip code for customer.
* Creation date: (02/11/02 10:31:32 AM)
* @return java.lang.String
*/
public java.lang.String getCustomerZipCode() {
return customerZipCode;
}
/**
* Sets address for customer.
```

```
* Creation date: (02/11/02 10:42:16 AM)
*/
public void setCustomerAddress(java.lang.String newCustomerAddress) {
customerAddress = newCustomerAddress;
}
/**
* Set city for customer.
* Creation date: (02/11/02 10:53:31 AM)
*/
public void setCustomerCity(java.lang.String newCustomerCity) {
customerCity = newCustomerCity;
}
/**
* Sets country for customer.
* Creation date: (02/11/02 10:57:06 AM)
*/
public void setCustomerCountry(java.lang.String newCustomerCountry) {
customerCountry = newCustomerCountry;
}
/**
* Sets first name of customer.
* Creation date: (02/11/02 10:57:31 AM)
*/
public void setCustomerFirstName(java.lang.String newCustomerFirstName) {
customerFirstName = newCustomerFirstName;
}
/**
* Sets internet address for customer.
* Creation date: (04/02/02 12:32:18 PM)
* @param newCustomerInternetAddress java.lang.String
*/
public void setCustomerInternetAddress(String newCustomerInternetAddress) {
customerInternetAddress = newCustomerInternetAddress;
}
/**
* Sets last name of customer.
* Creation date: (02/11/02 10:57:57 AM)
*/
public void setCustomerLastName(java.lang.String newCustomerLastName) {
customerLastName = newCustomerLastName;
}
/**
* Sets list of customers.
* Creation date: (02/11/02 10:58:12 AM)
*/
public void setCustomerListVector(Vector newCustomerListVector) {
customerListVector = newCustomerListVector;
}
```

```
/**
* Sets middle initial of customer.
* Creation date: (02/11/02 10:58:39 AM)
*/
public void setCustomerMiddleInitial(java.lang.String newCustomerMiddleInitial) {
customerMiddleInitial = newCustomerMiddleInitial;
}
/**
* Sets phone number for customer.
* Creation date: (02/11/02 10:58:39 AM)
*/
public void setCustomerPhoneNumber(java.lang.String newCustomerPhoneNumber) {
customerPhoneNumber = newCustomerPhoneNumber;
}
/**
* Sets state for customer.
* Creation date: (02/11/02 10:58:52 AM)
*/
public void setCustomerState(java.lang.String newCustomerState) {
customerState = newCustomerState;
}
/**
* Sets zip code for customer.
* Creation date: (02/11/02 10:59:08 AM)
*/
public void setCustomerZipCode(java.lang.String newCustomerZipCode) {
customerZipCode = newCustomerZipCode;
}
}
```

第 6 章 WebSphere(R) Application Server と Lotus(R) Domino(TM) との相互運用性の概要

iSeries(TM) システム・テストの flights シナリオは、 WebSphere Application Server によるフロントエンドと、バックエンドの Lotus Domino データベースで構成されています。そのため、WebSphere Application Server のフロントエンドと、 Lotus Domino のバックエンドが確実に連動できるようにする必要がありました。これには、2 つの環境間で通信する能力、シングル・サインオン (SSO) を両環境間で共有する能力、および各環境内でのセキュリティーを確保する能力が含まれます。

flights アプリケーションでは、両環境間での通信を行うために、 Java(TM) で記述された、Lotus Domino Java API を利用する WebSphere Application Server プログラムを使用しました。 Lotus Domino ワークフローが、フライトの取り込みを管理しました。 flights 社の顧客向け Web サイトのインプリメントには、WebSphere Application Server のトランザクション・サービスが使用されました。

flights アプリケーションでは、Lotus Domino LDAP (Lightweight Directory Access Protocol) ディレクトリーを使用します。 LDAP ディレクトリーは、WebSphere Application Server のセキュリティーのインプリメントに必要です。 LDAP を使用することで、flights 社の顧客は、顧客番号とパスワードを使用して認証を行うことができます。 flights アプリケーションは、ユーザーの ID を検証して、顧客情報および支払情報へのアクセスを許可します。

flights アプリケーションでは、WebSphere Application Server と Lotus Domino との間にわたるシングル・サインオン機能を使用します。これにより、ユーザーが一度サインオンするだけで、flights アプリケーションはその双方にアクセスできるようになります。具体的には、WebSphere Application Server から Lotus Domino データベースへのトランザクションが開始されるときに、同じログイン・クリデンシャルが使用されます。逆方向のトランザクションでも同じことが可能ですが、flights アプリケーションでは、Lotus Domino から WebSphere Application Server にアクセスする必要はありませんでした。シングル・サインオンは、複数の製品間にわたってシームレスな情報の流れを提供することを目的としています。

WebSphere(R) Application Server と Lotus(R) Domino(TM) との 相互運用シングル・サインオン

flights アプリケーションにおいて、セキュリティーは重要なポイントです。 WebSphere Application Server と Lotus Domino はいずれも、アクセスとデータの保護をサポートしています。両製品とも、ユーザー ID の判別および検証 (認証) と、指定されたユーザーに対する保護リソースへのアクセス権の付与 (許可) を伴う、セキュリティー機構をインプリメントしています。

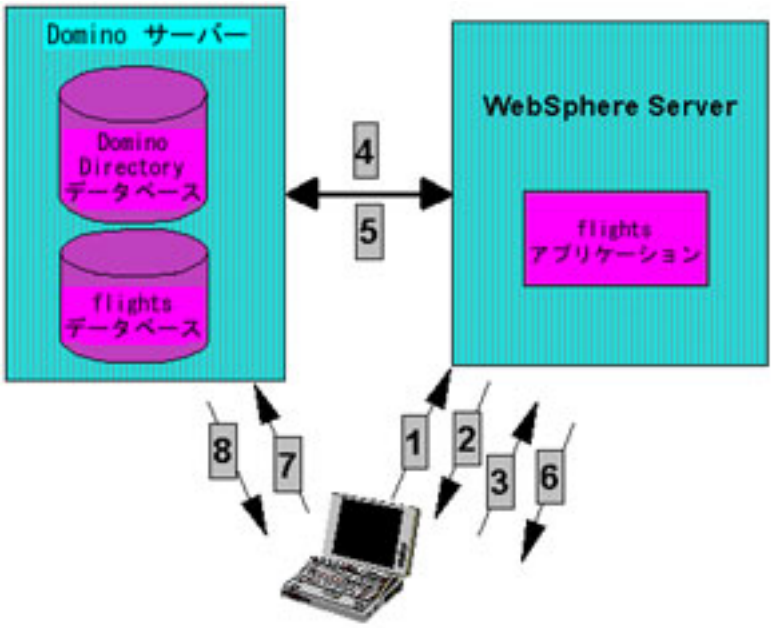
シングル・サインオン (SSO) を使用することで、 flights の Web ユーザーは WebSphere Application Server に対して一度認証を行い、その後は再度ログインする必要なしに Lotus Domino にアクセスできます。これは、Lotus Domino と WebSphere Application Server の両方で単一の Lotus Domino LDAP サーバーから取得した認証情報を共有するように構成することによって実現できます。

サーバー間での SSO を使用可能にするために、 Lightweight Third Party Authentication (LTPA) 機構を使用します。この機構では LTPAToken を使用します。LTPAToken には、ユーザー認証情報、SSO の有効範囲のネットワーク・ドメイン、および有効期限が含まれています。 LTPAToken は LTPA 鍵を使用して暗号化されます。この LTPA 鍵は、SSO に参加するサーバー間で共有する必要があります。

トークンは、Cookie として Web ユーザーに発行されます。この Cookie は、ブラウザーのメモリー内に常駐し、ユーザーがブラウザーを閉じると、ユーザーのコンピューター上には保管されずに消滅します。

WebSphere 上でシングル・サインオン (SSO) を使用可能にするには、 Enterprise Application Resource (EAR) ファイルをセキュリティー用に構成する必要があり、また WebSphere Application Server のグローバル・セキュリティー設定を構成する必要があります。図 1 に、flights アプリケーションでの、Lotus Domino と WebSphere Application Server との間のシングル・サインオンのプロセスを示します。

図 1 Lotus Domino と WebSphere の SSO



1. Web ユーザーが、ホーム・ページにアクセスするために、保護リソースに対する要求を Web サーバー (HTTP サーバー) に発信します。
2. Web サーバーが、ユーザーに認証情報の入力を求めるプロンプトを出します。
3. ユーザーは、情報 (顧客番号とパスワード) を入力して応答します。
4. 次に Web サーバーは、LTPA サーバー (WebSphere Application Server) と通信し、 LTPA サーバーは Lotus Domino Directory に接続して認証情報を検証します。
5. ユーザーの提供した情報が正しい場合、Lotus Domino は、サーバー (WebSphere Application Server) に対し、検証された情報を返します。
6. サーバーは戻された値を使用して、要求されたリソースに対するアクセス権をそのユーザーが有しているかどうかをチェックし、ユーザーに LTPA トークンを発行します。 Web サーバーは、トークンを HTTP Cookie としてユーザーに送信します。この Cookie はユーザーのブラウザーに保管され、要求されたリソース (index.html) を提供します。

7. ユーザーが認証されて Cookie が使用可能になると、そのユーザーは Lotus Domino または WebSphere Application Server にある他の保護リソースを要求できます。
8. Lotus Domino および WebSphere Application Server は、ユーザーについて提供されたトークンを検証し、ユーザーがそのリソースに対する適切なアクセス権を有している限り、再びユーザー確認の情報を求めるプロンプトを出さずに、要求されたリソースをブラウザーに送信するよう Web サーバーに指示します。

Lotus Domino 上での IIOP のセットアップ

Lotus Domino のリモート・オブジェクトは、 Common Object Request Broker Architecture (CORBA) を使用して、 Lotus Domino へのアクセスを行います。 CORBA では、オブジェクト間の通信は Object Request Broker (ORB) を介して行われます。 ORB は Internet Inter-ORB Protocol (IIOP) を使用してメッセージを相互にやり取りします。 Lotus Domino では、 Lotus Domino Internet Inter-ORB Protocol (DIIOP) サービスが CORBA 通信に使用されます。リモート・オブジェクトを使用すると、 Web アプリケーション・サーバーを Lotus Domino サーバーとは別のマシン上で実行できる、という利点があります。コードによって lotus.domino.* パッケージをインポートします。 NotesFactory() メソッドは、Lotus Domino サーバーの IP アドレスまたは名前を必要とし、また IIOP と Lotus Domino サーバーへのアクセス権を持つ有効なユーザー ID とユーザー・プロファイルを必要とします。リモート Lotus Domino サーバーで DIIOP が構成されている必要があり、またユーザー ID は、その Lotus Domino サーバーの使用、および IIOP エージェントの実行が許可されている必要があります。

flights アプリケーションでは、以下の理由から、 Lotus Domino Java(TM) API を使用する際にリモート Lotus Domino オブジェクトを使用しました。

- 構成および実行時の要件がより単純である。
- マルチスレッド環境での使用に制限がない。

Lotus Domino サーバーを CORBA 用に構成するには、以下の手順を実行します。

1. notes.ini ファイルを編集し、タスクを ServerTasks パラメーターで指定されているタスク・リストに追加する。例えば、次のようになります。
   ServerTasks=<その他の任意のタスク>, HTTP, DIIOP
2. Lotus Domino サーバー文書を編集用に開き、「ポート (Ports)」->「インターネット・ポート (Internet Ports)」->「IIOP」タブを選択する。ポート番号を設定し、ポートを使用可能にします。デフォルトのポート番号は、TCP/IP IIOP ポートの場合は 63149、SSL を使用する IIOP ポートの場合は 63148 です。また、名前とパスワードによるアクセス、および匿名アクセスを許可するかどうかを指定できます。
3. スレッド数とタイムアウト値を構成する。タイムアウト値は、アイドル状態の接続が、何分間後にサーバーによって除去されるかを表します。タイムアウト値を設定するには、 Lotus Domino サーバー文書の「インターネット・プロトコル (Internet Protocols)」->「IIOP」タブを選択します。
4. Lotus Domino サーバーへのアクセスと、そのサーバー上での Java プログラムの実行に関するセキュリティーを設定する。「サーバー・アクセス (Server access)」セクションで、サーバーにアクセスできるユーザーを指定します。フィールドを空白のままにすると、いかなるユーザーもサーバーへのアクセスを拒否されません。このフィールドにユーザーの名前がリストされている場合は、ここに明確にリストされたユーザーおよびグループのみが、サーバーにアクセスできます。ユーザーにログインが求められる場合には、ユーザーが認証された後で、この情報がサーバーによりチェックされます。
5. Lotus Domino サーバー文書の「Java/COM 制限 (Java/COM Restrictions)」セクションで、 CORBA を使用した Lotus Domino オブジェクトへのアクセスを許可するユーザーを指定する。このフィールドを

空白のままにすると、いかなるユーザーも、CORBA を使用した Lotus Domino オブジェクトへのアクセスを許可されません。このフィールドでは、ワイルドカードを使用することができます。このフィールドにアスタリスク (*) を入れると、すべてのユーザーが許可されます。

6. 使用する Lotus Domino サーバーがファイアウォールで保護されている場合は、 Lotus Domino サーバーの Notes(TM).ini を編集して、 DIIOP_IOR_HOST=ipAddress という行を追加してください。ここで、ipAddress は、ファイアウォールの外部から見た場合の、Lotus Domino サーバーの IP アドレスです。

flights アプリケーションのセキュリティーに関する構成

WebSphere Application Server または Lotus Domino の構成を開始する前に、アプリケーションのセキュリティーを構成して、インストールしておく必要があります。アプリケーションのセキュリティーの構成に関する情報については、 IBM(R) Redbook「IBM WebSphere V4.0 Advanced Edition Security」を参照してください。 WebSphere Studio Application Developer で flights アプリケーションのセキュリティーを保護する際には、 『Securing Web Components』および 『Securing EJBs』の章に従いました。
flights アプリケーションでは、フォーム・ベースの認証を使用しています。この認証機構により、 flights 社のサイトでは、サイト固有の HTML ログイン・ページを指定することが可能となりました。フォーム・ベースの認証を使用する場合には、パスワードは暗号化されず、またターゲット・サーバーも認証されないため、セキュリティー上のリスクが存在します。このセキュリティー上のリスクを避けるには、セキュア・トランスポート (SSL) を使用することができます。

WebSphere Application Server でのシングル・サインオンの使用可能化

WebSphere 上で、SSO 用にグローバル・セキュリティー設定を構成するには、以下の手順を実行します。

1. WebSphere 管理コンソールを開始する。
2. 「コンソール (Console)」->「セキュリティー・センター (Security Center)」を選択する。 WebSphere のグローバル・セキュリティー設定が表示されます。「一般 (General)」タブの「セキュリティーを使用可能にする (Enable Security)」にチェックマークを付けます。
3. 「認証 (Authentication)」タブをクリックし、認証機構のタイプとして「Lightweight Third Party Authentication (LTPA)」を選択する。
4. LTPA 設定を以下のように指定する。
   a. 「トークン有効期限 (Token Expiration)」フィールドで、 LTPA トークンを使用するクライアントが再度認証しなければならなくなるまでの時間を指定する。
   b. 「シングル・サインオン (SSO) の使用可能化 (Enabled single sign-on (SSO))」にチェックマークを付ける。「ドメイン (Domain)」フィールドが使用可能になります。
   c. 「ドメイン (Domain)」フィールドに DNS のドメイン名を入力する。このドメイン名は SSO 用の HTTP Cookie の作成時に使用され、SSO が適用される有効範囲を決定します。

重要: SSO に参加するサーバーはすべて、同じ DNS ドメイン内に存在していなければなりません。

1. 「LDAP」ラジオ・ボタンを選択状態にして、LDAP サーバー設定を入力する。

注: 必ず、セキュリティー・センターで、Lotus Domino 5.0 をディレクトリー・タイプとして使用するように WebSphere Application Server を構成してください。また、Lotus Domino サーバーが稼働しており、LDAP タスクが開始していることを確認してください。これは、セキュリティー・サーバーの ID とパスワードが検査されるためです。

1. 「鍵の生成 (Generate Keys)」ボタンをクリックし、LTPA トークンを暗号化するための LTPA 鍵を作成する。暗号鍵のセットを保護するための LTPA パスワードの入力を求めるプロンプトが出されます。これらの LTPA 鍵は、SSO を使用するすべてのサーバー間で共有する必要があります。

重要: LTPA 鍵の生成は、Lotus Domino LDAP サーバー設定を構成する際に行う必要があることに注意してください。これにより、LDAP ホスト名およびポートを、エクスポート・ファイル内に確実に含めることができます。 Lotus Domino は、Web SSO 構成文書の作成処理時に、この情報を必要とします。

1. LTPA 鍵が生成されたら、「鍵をエクスポート (Export Key)」ボタンをクリックして、LTPA 鍵をファイルにエクスポートする。このファイルは、Lotus Domino に鍵をインポートするために使用されます。
2. 「適用 (Apply)」をクリックして、「OK」をクリックする。
3. プロセスが完了すると、「管理サーバーが再始動されるまで、変更内容は有効になりません。(Changes will not take effect until the admin server is restarted.)」という警告メッセージが表示される。「OK」をクリックします。
4. コンソールの左側のツリー表示で、ノード・フォルダー内のノードを選択して右クリックし、表示されたコンテキスト・メニューで「再始動 (Restart)」を選択して、管理サーバーを再始動する。

Lotus Domino サーバーでのシングル・サインオンの使用可能化

1. Lotus Domino Directory データベース内に、新規の Web SSO 構成文書を作成する。
   a. 「サーバー (Server)」->「サーバー (Servers)」を選択して、ビューを表示する。「Web」ボタンをクリックし、表示されたコンテキスト・メニューで、「Web SSO 文書を作成 (Create Web SSO Document)」を選択します。
   b. 「(LTPAToken)」という「トークン名 (Token Name)」フィールドを持つ新規文書が表示される。
   c. 「トークン・ドメイン (Token Domain)」フィールドに、DNS ドメインを入力する。この値は、WebSphere Application Server の「ドメイン (Domain)」フィールドで指定した値と一致しなければなりません。このドメイン名は、シングル・サインオン用の HTTP Cookie の作成時に使用され、シングル・サインオンが適用される有効範囲を決定します。
   d. SSO シナリオに参加する Lotus Domino サーバーを選択する。

   注: Lotus Domino サーバーの完全修飾サーバー名を指定する必要があります (例: MyDominoServer/MyOu)。指定する Lotus Domino サーバー名は、使用する Lotus Notes(R) クライアントの、アクティブなロケーション文書に現在含まれている、ホーム/メール・サーバーの名前とも一致しなければなりません。 Location 文書が一致しない場合は、一致するものを作成しなければなりません。

   a. 発行されたトークンを最大で何分間有効とするかを、「有効期限 (分)(Expiration (minutes))」ファイルに入力する。 WebSphere Application Server で設定した時間と一致するように設定してください。
   b. 「鍵 (Keys)」ドロップダウンをクリックし、「WebSphere LTPA 鍵をインポート (Import WebSphere LTPA keys)」を選択する。
   c. 先ほどエクスポートした WebSphere Application Server LTPA 鍵ファイルの、パスおよびファイル名を指定する。
   d. 「OK」をクリックする。新しいダイアログ・ボックスが表示され、鍵の生成時に指定した LTPA パスワードの入力をユーザーに求めます。
   e. 「OK」をクリックする。処理が完了すると、確認のメッセージが表示されます。
   f. 文書中に、新たに「WebSphere 情報 (WebSphere Information)」セクションが表示される。 LDAP レルムおよびホスト名は、WebSphere Application Server の Import ファイルから読み取られます。WebSphere LDAP 構成の設定でポートが指定されている場合は、「LDAP レルム (LDAP Realm)」フィールドで、コロン (:) の前に必ず円記号 (¥) を追加してください。

g. 「保管 (Save)」ボタンをクリックし、「閉じる (Close)」ボタンをクリックする。文書が保管されます。 Lotus Domino Directory 内に文書が存在するかどうかを確認するには、「サーバー (Server)」->「Web 構成ビュー (Web Configurations View)」を選択し、「*- 全てのサーバー - (*-All Servers -)」セクションを展開します。新規文書は、「LtpaToken 用 Web SSO 構成 (Web SSO configuration for LtpaToken)」として表示されます。

2. 「ポート (Ports)」->「インターネット・ポート (Internet Ports)」->「Web」タブで TCP/IP ポート状況を使用可能にする。 Lotus Domino サーバー文書を変更することによって、TCP/IP を使用した匿名の接続を許可してはなりません。
3. サーバー文書の「インターネット・プロトコル (Internet Protocols)」->「Lotus Domino Web エンジン (Lotus Domino Web Engine)」タブで、「マルチサーバー・セッション (Multi-Server session)」を選択する。
4. 「セキュリティー (security)」タブの「Web サーバー認証 (Web server authentication)」セクションで、「多種類の名前 (低セキュリティー) (More name variations with lower security)」を選択する。これにより、ユーザーは、名前とパスワードのダイアログ・ボックスに、姓、名、通称、完全な階層名、短縮名、およびエイリアス名などの名前形式で入力できるようになります。

   表 1 に、flights アプリケーションのデータベースでセキュリティーをインプリメントするために、データベース ACL がどのように変更されたのかを示します。

表 1 Lotus Domino と WebSphere の SSO

人物、サーバー、およびグループ	ユーザー・タイプ	アクセス・レベル	権限
flights 社従業員	Person Group	Editor	文書の削除、Lotus Script/Java エージェントの作成
Anonymous	Unspecified	Reader	共用文書の書き込み

ユーザーがデータベースから Web ページをアクセスしようとすると、 Lotus Domino サーバーは、デフォルトのサーバー・ログイン・ページを表示します。 Web ユーザーは、自分のユーザー名とパスワードを入力して、「ログイン (Login)」ボタンをクリックしなければなりません。

Lotus Domino サーバーは、ユーザーが Lotus Domino Directory のデータベースに登録されているかどうかを確認し、クリデンシャルの値が正しいことを検証します。また、サーバーは、ユーザーがデータベースへのアクセス権を有しているかどうかをチェックします。ユーザーが認証されると、 Lotus Domino は新規の LTPAToken を作成し、 HTTP Cookie としてユーザーに送信して、Lotus Domino 文書を開きます。

WebSphere(R) Application Server と Lotus(R) Domino(TM) の相互運用性に関する主要な発見事項

ここでは、WebSphere Application Server と Lotus Domino の相互運用性を扱う flights シナリオのインプリメントを通して得られた、主要な発見事項について記載します。

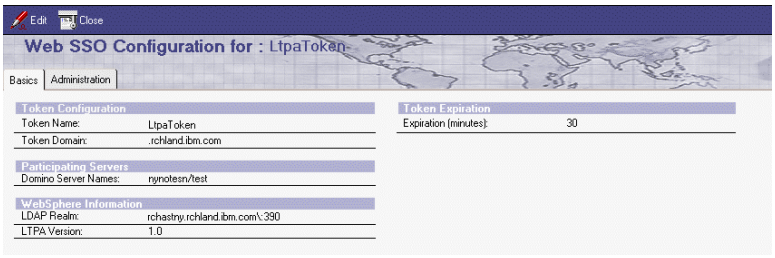
- Lotus Domino の Web シングル・サインオン (SSO) 構成文書を作成するときには、使用する Notes(TM) クライアントのロケーション文書が、SSO を使用可能にする Lotus Domino サーバーをポイントするよ

うにしなければなりません。これは、サーバー用に公開鍵を使用できるようにするために必要です。Web SSO 構成文書を保管するときに、サーバーが見つからないというメッセージが表示される場合は、上記の方法によって解決することができます。

- WebSphere Application Server セキュリティー・センターでは、レルム (ドメイン名) を小文字で指定する必要があります。

- WebSphere Application Server は DNS 名の大文字と小文字を区別することから、 Lotus Domino で DNS ドメインの値を使用するときには、図 1 のように、この値を大文字小文字も含めて完全に同じになるように指定してください。

図 1 Lotus Domino と WebSphere の SSO



- セキュリティーおよび SSO が構成されている場合、Lotus Domino または WebSphere のいずれかの URL にアクセスするときには、 URL を、ドメイン名付きで完全修飾する必要があります (例えば、http://*systemName.domainName:portNumber/uri*)。 URL を完全修飾していない場合 (例えば、http://*systemName:portNumber/uri*)、ログイン・フォームに戻され、アクセスしようとしたページには移動できません。

- WebSphere Application Server でセキュリティーを構成する場合は、 NCSOW.jar を qibm/userdata/webasadv4/*instanceName*/lib/ext ディレクトリーに入れます。 NCSOW.jar ファイルを使用して Lotus Domino セッションを作成していたため、当初、この jar ファイルをデフォルトのサーバーの JVM プロパティーに入れました。しかし、これによって、セキュリティーが使用可能になったあとで問題が発生しました。 jar ファイルが JVM 設定に入っていたので、com.ibm.ejs.oa.EJSORB に対して NoClassDefFound が発生し続ける結果となりました。この問題は、NCSOW が JVM クラスパスにあったために発生していました。問題を解決するために、NCSOW.jar を qibm/userdata/webasadv4/*instanceName*/lib/ext ディレクトリーに移動し、 JVM クラスパスから項目を除去しました。

- セキュリティー使用時には、 Context Enterprise Bean の getCalledIdentity() メソッドを使用することで、正しい ID のもとで実行しているかどうかを検証できます。このメソッドは、Enterprise Bean 内のメソッドの実行に使用されているユーザー ID を表示します。

- 以下に示す Java/CORBA クラス (lotus.domino パッケージ) の要素は、シングル・サインオン・ドメイン内の Lotus Domino サーバーと WebSphere サーバーへのサインオンをサポートします。

– SessionToken プロパティー
読み取り専用。シングル・サインオンをサポートするドメイン内の Lotus Domino サーバーと WebSphere サーバーにサインオンできるようにするためのセッション・トークンを取得します。
注: このプロパティーは、R5.0.5 でインプリメントされた新規プロパティーです。
定義場所: lotus.domino.Session
データ型: String
構文: public String getSessionToken() throws NotesException
使用法: トークンは、各ユーザーに固有であり、その有効期間は Lotus Domino Directory 内で指定されています。トークンのフォーマットは、WebSphere で使用される LtpaToken Cookie と一貫性を持ちます。

また、HttpServletRequest.getCookies() を使用して、サーブレットの HTTP ヘッダーからトークンを取得することもできます。
このプロパティーは、シングル・サインオン用に構成されたサーバー上でのみ有効です。
使用法および例については、NotesFactory を参照してください。

– NotesFactory クラス
注: WebSphere 環境で Lotus Domino オブジェクトに対してリモート (IIOP) 呼び出しを行うには、NCSOW.jar がクラスパスに含まれている必要があります。これは、R5.0.4 からの変更点です。
NotesFactory クラスの記述は、以下のように拡張されました。
注: これらの拡張は、R5.0.5 で新たに加えられたものです。

シングル・サインオンを使用してサーバーにアクセスするには、以下のようにして、Session オブジェクトを作成します。リモート (IIOP) 呼び出しの場合、最初のパラメーターは、ホストのインターネット名です。ローカル・コールの場合、最初のパラメーターはヌルです。

- createSession(hostString, String token) - アクセスは、トークンに基づいて許可されます。このメソッドは、Lotus Domino 環境で動作します。トークンは、Session.getSessionToken か、WebSphere の使用する LtpaToken Cookie のいずれかより取得した、シングル・サインオン用の有効なトークンでなければなりません。
- createSession(hostString, org.omg.SecurityLevel2.Credentials) - アクセスは、クリデンシャル・オブジェクトに基づきます。このメソッドは、loginHelper を使用してクリデンシャル・オブジェクトが作成される、WebSphere 環境で動作します。
- createSession(hostString, null) - WebSphere 環境内の現行のクリデンシャル・オブジェクトに基づいて、アクセスが許可されます。このメソッドは、WebSphere 内の Enterprise Bean アプリケーションから動作させることができます。

NotesFactory の仕様は、以下のメソッドを含むように拡張されました。

- static public Session createSession(String host, String token) throws NotesException
- static public Session createSession(String host, org.omg.SecurityLevel2.Credentials) throws NotesException

– 例
- **例 1:** この Lotus Domino エージェントは、シングル・サインオン用のトークンを取得し、トークンに基づいて他のサーバーへのリモート (IIOP) セッションを作成します。

```
import lotus.domino.*;
public class JavaAgent extends AgentBase {
```

```
public void NotesMain() {
try {
Session session = getSession();
AgentContext agentContext = session.getAgentContext();
Session s2 = NotesFactory.createSession("test5.iris.com",
session.getSessionToken());
System.out.println("remote session name = " + s2.getUserName());
} catch(Exception e) {
e.printStackTrace();
}
}
}
```

- **例 2:** このサーブレットは HttpServletRequest を介して、 LTPAToken Cookie からシングル・サインオン用のトークンを取得し、トークンに基づいてセッションを作成します。

```
import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import lotus.domino.*;


public class Cookies extends HttpServlet
{


private void respond(HttpServletResponse response, String entity) throws IOException
{
response.setContentType("text/plain");
if (entity == null)
{ response.setContentLength(0);}
else {
response.setContentLength(entity.length() + 1);
ServletOutputStream out = response.getOutputStream();
out.println(entity);
}
}


public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
String s1 = "";
Cookie[] cookies = null;
String sessionToken = null;
```

```
try {
cookies = request.getCookies();
}
catch (Exception e) {
respond(response,“Exception from request.getCookies(): ” + e.toString());
return;
}


if (cookies == null) {
s1 = “No cookies received”;
}
else {
for (int i = 0; i < cookies.length; i++) {
if (cookies[i].getName().equals(“LtpaToken”)) {
sessionToken = cookies[i].getValue();
}
}
}


if (sessionToken != null) {
try {
NotesThread.sinitThread();
Session session = NotesFactory.createSession(null, sessionToken);
s1 += “¥n” + “Server: ” + session.getServerName();
s1 += “¥n” + “IsOnServer: ” + session.isOnServer();
s1 += “¥n” + “CommonUserName: ” + session.getCommonUserName();
s1 += “¥n” + “UserName: ” + session.getUserName();
s1 += “¥n” + “NotesVersion: ” + session.getNotesVersion();
s1 += “¥n” + “Platform: ” + session.getPlatform();
NotesThread.stermThread();
}
catch (NotesException e) {
s1 += “¥n” + e.id + e.text;
e.printStackTrace();
}
}


respond(response,s1);
}
}
```

- **例 3:** このアプリケーションの断片は、WebSphere から取得したクリデンシャル・オブジェクトに基づき、セッションを作成します。

```
com.ibm.CORBA.iiop.ORB orb = com.ibm.ejs.oa.EJSORB.getORBInstance();
```

```
if (orb != null) {
org.omg.SecurityLevel2.Current(R) securityCurrent =
(org.omg.SecurityLevel2.Current)orb.resolve_initial_references("SecurityCurrent");
org.omg.SecurityLevel2.Credentials invCred =
securityCurrent.get_credentials(org.omg.Security.CredentialType.SecInvocationCredentials);
System.out.println("creating notes session using current creds");
session = NotesFactory.createSession(notesServer, invCred);
}
```

- **例 4:** この WebSphere Enterprise Bean アプリケーションは、WebSphere 環境内の現行のクリデンシャル・オブジェクトに基づいて、セッションを作成します。

```
import lotus.domino.*;

public class HelloBean extends Object implements SessionBean {

... /* See HelloBean.java from Websphere for the complete class code */

/**
Returns the greeting. But has been modified to create a remote session to the
Lotus Domino server.
@return The greeting.
@exception RemoteException Thrown if the remote method call fails.
*/
public String getMessage () throws RemoteException
{
String result = "hello bean ";

try {
Session s = NotesFactory.createSession("test5.iris.com", null);
result = result + " — Got Session for " + s.getUserName();
}
catch (NotesException ne) {
result = result + "— " + ne.text;
result = result + "— failed to get session for user";
}

return (String) result + " — done";
}
}
```

第 7 章 参照

以下のリソースには、有用な情報が記載されています。

- 「IBM(R) WebSphere V4.0 Advanced Edition Security」IBM Redbook (SG24-6520-00)
- 「Lotus Domino and WebSphere Integration on the IBM eServer iSeries(TM) Server」IBM Redbook (SG24-6223)
- 「Lotus Domino and WebSphere Together Second Edition」IBM Redbook (SG24-5955-01)
- 「セキュリティー・ガイド (Security Guide)」
  http://www.ibm.com/software/webservers/appserv/doc/v40/aes/infocenter/was/pdf/nav_Securityguide.pdf
- WebSphere の相互運用性
  http://www-106.ibm.com/developerworks/webservices/library/ws-bpinter/

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12