Java™ magazine

By and for the Java community

Java EE 7: LICENSE TO CODE

生産性を改善し、次世代のWebアプリを構築

11 JAVA: ESPN.COMでのスラムダンク

15 銀行業務におけるJAVA

72 INSTAGRAMアプリの構築

ORACLE®

//table of contents /

COMMUNITY

02
編集長より

03
Java Nation
ニュース、人、イベント

07
JCP Executiveシリーズ
Mark Littleに聞く
Red Hatのエンジニアリング部門担当バイス・プレジデント、JCPとJava EE 7を語る

JAVA IN ACTION

11
スラムダンク
Java、ESPN.comを勝者に

15
銀行業務におけるJava
Java EEベースのシステム、ブラジルの複雑な金融市場に安定性をもたらす

JAVA TECH

27
Javaアーキテクト
invokedynamicの神秘を解く
Julien Pongeによるinvokedynamicシリーズのパート2

33
Javaアーキテクト
ラムダ式を使用した高度な操作
Ben EvansとMartijn Verburgによるシリーズ完結編

37
Javaアーキテクト
開発者をクラウドへと駆り立てるものとは
クラウド入門書

58
リッチ・クライアント
JavaFX Scene Builderのエンタープライズ・アプリケーションへの統合
Scene BuilderのAdam Bien

66
モバイルと組込み
通信ネットワークの構築
標準ベースのAPIで通信サービスを構築する

72
モバイルと組込み
Instagram Java ME アプリケーションの構築
Vikram Goyalがその方法を解説する

76
Fix This
コードの問題に挑戦

表紙画像：I-HUA CHEN

19
Java入門
JAVA EE 7を探索する
Max Bonbhel、Java EE 7の新規APIの使用方法について解説する

40
JAVA EE 7の到来
オラクルのAnil GaurがJava EEの進化を語る

46
エンタープライズJava
開発生産性の向上
Java EE 7の仕様拡張と新規APIを利用して生産性を向上する

51
エンタープライズJava
HTML5の活用
Java EE 7で次世代アプリを作成するための新機能を探究する

//from the editor /

Caroline Kvitka
（@oraclejavamag）：2001 年にオラクルに入社し、Oracle Magazine、Profit、Java Magazine の編集長を務める。 90 年代の初めからテクノロジー関連の記者を務め、複数のテクノロジーおよび E コマース事業の立ち上げに参加した経験を持つ。

待ち望んでいたものがついにやってきました。 開発者の生産性を高め、Web アプリケーションに新しい能力を加えることができる数多くの新ツールとともに、Java EE 7 がリリースされました。 読者のみなさんが私と同じ性格なら、何か新しいものが玄関に届くとすぐさま箱を開けて使ってみたいと思うことでしょう。 今号の Java Magazine では、Java EE 7 の使用開始に役立つ情報をお届けします。是非、実際にお試しください（今号ではすべての Java EE 関連コンテンツに Java EE 7: License to Code アイコンを表示しています）。

はじめに、オラクルのソフトウェア開発担当 Vice President である Anil Gaur がインタビューの中で、HTML5 サポート、WebSocket、JAX-RS、JSON 処理、バッチ処理、並列処理などの、注目すべき Java EE 7 の新機能に関する重要な事項について説明します。 次に、Max Bonbhel 氏が Java EE 7 の新しい API について取り上げ、詳しく説明します。これは大きなテーマであるため、今後数回にわたる連載としてお届けします。 さらに、Johan Vos 氏が新リリースでの生産性の向上について探り、Java EE 7 によっていかにデフォルトの開発・構成が簡略化されるかを示します。 そして最後に、オラクルの Java カリキュラム開発者である Eduardo Moranchel と Edgar Martinez が、次世代のインターネット・アプリケーション開発を可能にする Java EE の新機能として、WebSocket、HTML5、および JSON を紹介します。

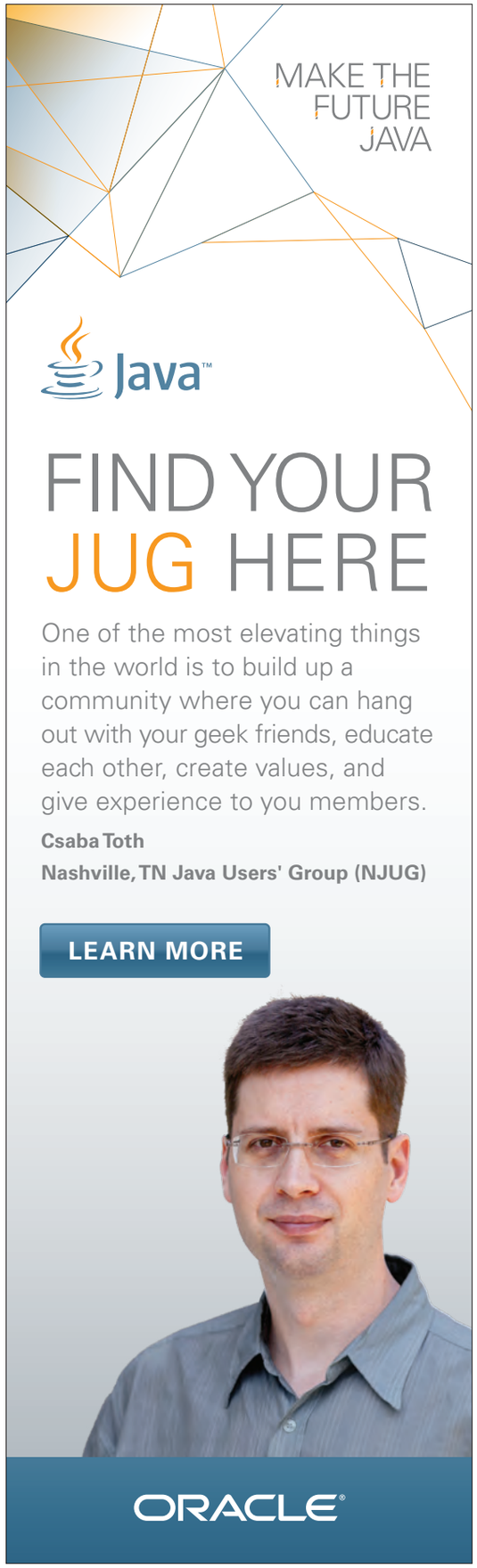
その他に、ESPN でのスポーツ・ファンへの最新情報の提供から、ブラジルでの複雑な銀行業務課題の解決まで、実際に Java EE が使用されている事例を取り上げます。

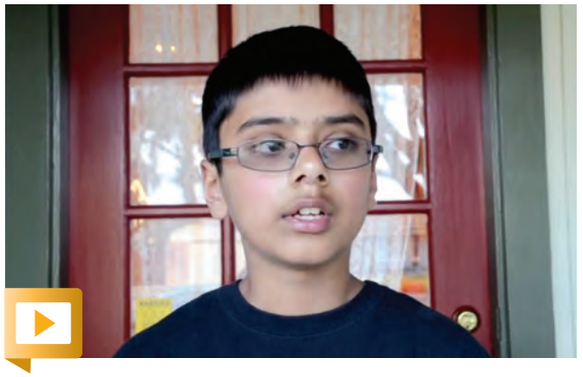
読者のみなさんと共有していただける Java の使用例や Java ユーザー・グループでの実績をお持ちの場合、または、優れたコーディング技術を最終ページの Fix This コラムで紹介したい場合や、それ以外のフィードバックがある場合は、編集部までお送りください。

それでは Java EE 7 を満喫してください。

編集長 Caroline Kvitka BIO

写真：BOB ADLER





ワークショップについて自身の考えを語る Aditya Gupta。

MINECRAFT を使用した JAVA ワークショップ

左：説明を聞く Minecraft ファンの子供達、右：車や果物のたとえを用いて Java プログラミングを教える Arun Gupta。

写真：FRED WORLEY、MENKA GUPTA

2013 年 3 月 16 日、Minecraft を使用した Java プログラミングを学ぶため、オラクルの Java エバンジェリストである Arun Gupta の自宅に 10 人の子供達が集まりました。 小中学生に大人気のこのゲームでは、3D の世界に、テクスチャの付いた立方体で構造物を造ることができます。 このゲームは Java で書かれており、"MOD"（modification の略）を作成して追加していくことができます。 最近、息子の休日に一緒に MOD を作成して過ごした Gupta は、Minecraft ファンの子供達をコーディング・ワークショップに招くことに決めました。 10 歳から 14 歳までの参加者のほとんどが Java の経験はもちろんのこと、プログラミングの経験もありませんでした。 しかし、彼らには豊富な Minecraft 経験がありました。 ゲームに対する子供達の熱意を利用して、Gupta は Java プログラミングを紹介しました。 Gupta は車や果物についての分かりやすい概念を使用して、クラス、プロパティ、メソッド、インタフェース、例外といった中心的概念を説明しました。 NetBeans を使用した Hello World アプリケーションを初めて作成し、実行したワークショップ参加者に一言で感想を求めると、「楽しい」、「簡単」、「すぐできる」、「すごい」、「短い」、「直感的」という答えが返ってきました。 セッションが終わる頃には、MOD の作成フレームワーク全体が完成しました。 ワークショップで作成した MOD は、新しいサーバー・サイド・コマンドを 1 つ追加して簡単なメッセージを出力するものでしたが、参加者のうれしそうな顔はかけがえのないものでした。 あなたも Minecraft ワークショップを開催しませんか。 詳細はこちら。

OpenJDK 運営委員会、一般代表メンバーを選出

OpenJDK コミュニティの構成や運営を監督する OpenJDK 運営委員会は、2 名の一般代表メンバーとして **Andrew Haley** 氏と **Doug Lea** 氏を選出しました。 両氏は以前より一般代表メンバーとして参加していましたが、4 月 1 日に新たな 1 年の任期が始まりました。

Red Hat に勤める Haley 氏は候補者声明文において、「自由を守ること、また、仕事を成し遂げるために大量のコードを作成する人々の役に立つ方向へ運営委員会を導くこと」という目標を掲げました。

SUNY Oswego に所属する Lea 氏の目標は、「OpenJDK のプロセスとメカニズム継続的な改善を提唱する役割を持続することで、特に学校、研究機関、個人貢献者のコミュニティに良い影響を与えること」です。

JAVA CHAMPION のご紹介

ANTONIO GONCALVES

Antonio Goncalves 氏はシニア・ソフトウェア・アーキテクト兼ライターであり、パリ Java **ユーザー・グループおよび** Devoxx France **の共同創設者です。多数の** Java EE JSR **の** Expert Group **メンバーとして** Java Community Process（JCP）**に積極的に参加しています。** 2009 **年** 6 **月に、**Java Champion **となりました。**

Java Magazine：ご出身はどちらですか。

Goncalves：パリ育ちですがポルトガル人なので、生活にはいつも 2 つの国、文化、言葉があります。

Java Magazine：コンピュータやプログラミングに興味を持つようになったのはいつのことですか。

Goncalves：12 歳のときです。学校に Thomson TO7 と MO5（Motorola 6809 プロセッサ搭載）があって、BASIC コースを学習していました。

Java Magazine：最初のコンピュータと、最初に習得したプログラミング言語を教えてください。

Goncalves：Commodore 64 と、BASIC およびアセンブリ言語です。 当時の Commodore Magazine には、ゲーム開発用のコードが何十ページも掲載されていました。私は実際に何週間もかけてコードを書いてはテープに保存し、実行し、デバッグしていました。 その頃の私には GOTO が素晴らしいキーワードでした。

Java Magazine：仕事として最初に行ったプログラミング作業を教えてください。

Goncalves：COBOL を使用したバッチの作成です。 Commodore 64 でゲーム用のプログラムを作り続けられればよかったのですが、仕事をして生活していくために必要な言語は COBOL でしたから。 その後で C++ や Smalltalk、それから Visual Basic を少し。 Java と出会ったのは 1998 年です。

Java Magazine：リラックスするためにどんなことをしていますか。

Goncalves：2 年間にわたって美術史を学んだ経験がありますし、住んでいる場所はパリですから、あらゆる文化的資産を利用しています。 ジャズ・コンサートや美術展にもよく行きます。

Java Magazine：お休みの日にはどんなことをしますか。

Goncalves：コードやブログを書いたり、マニア的なことをしたりしています。 8 歳の娘がいるので、コンサートや展覧会に行ったり、日帰りでサイクリングに出かけたりもします。

Java Magazine：仕事について、実際の業務以外でもっとも楽しいと思われるのはどんなことですか。

Goncalves：非常に孤立した仕事なので、実際に人に会うことが気分転換になります。

Java Magazine：Java チャンピオンになったことで日常生活は変わりましたか。

Goncalves：優れた才能を持つ人々との出会いが増えました。

Java Magazine：Java EE 7 がもうすぐリリースされますね。 この新リリースでもっとも重要な点について教えてください。

Goncalves：JMS（Java Message Service）2.0、JTA（Java Transaction API）1.2、JAX-RS 2.0 も新しくなっていますが、このリリースでの重要な動きは CDI（Contexts and Dependency Injection）との統合です。 ますます多くの Java EE 7 仕様が CDI を採用し始めているので、これによってプラットフォームに一体性がもたらされるでしょう。

Java Magazine：今後楽しみにしていることは何ですか。

Goncalves：意外なことに出会いたいですね。 私のキャリアには多くの変化がありましたが、これらの変化は私が目指したものによってもたらされた場合もあれば、予期せぬ出来事によってもたらされた場合もありました。 ジョン・レノンの言葉にあるように、「あれこれ別のことを計画しているうちに、身に降りかかってくることこそが人生」なのです。

詳しくは、AntonioGoncalves.org または twitter をご覧ください。

JVM 向けの言語 Golo のデモを実施

Golo が Devoxx France 2013 で公開されました。 暗黙的な指定よりも明示的な指定を好むこの言語は、数日と言わず数時間で習得可能です。 Golo は INSA-Lyon で、CITI Laboratory の DynaMid リサーチ・プロジェクトのメンバーによって開発されました。

最初から invokedynamic を使用して構築され、現在ベータ版で JVM の最新技術を取り入れています。 また、invokedynamic を使用した言語ランタイムを作成する方法の見本でもあります。

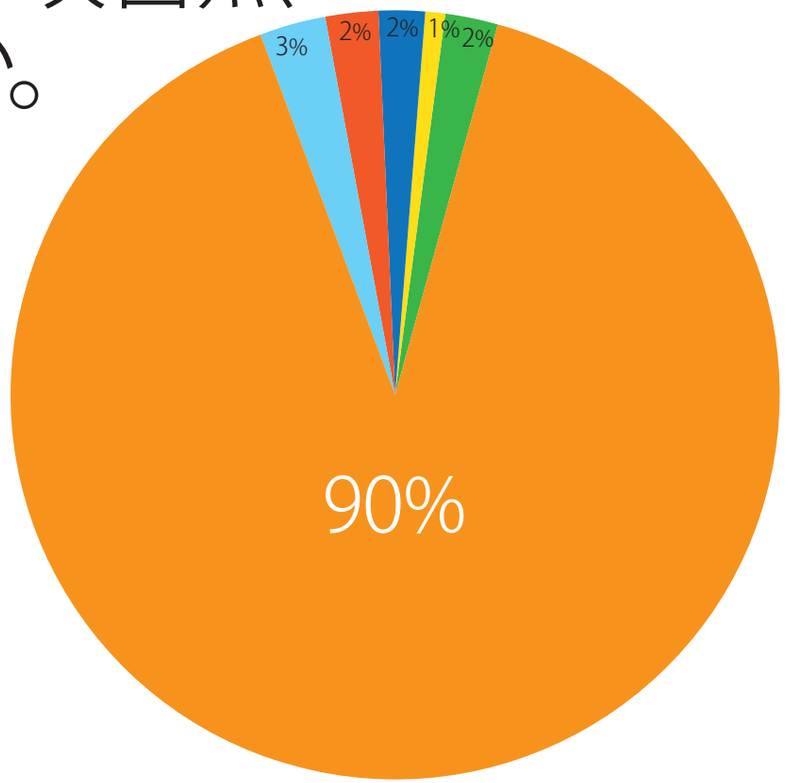
Golo を開発した **Julien Ponge** によると、JVM 向けに使用できる言語が数多くあるにもかかわらず、新たな言語の開発を決めた理由は 2 つあります。

「私は動的なミドルウェアおよびアプリケーションを中心に開発している少人数のチームで活動しています。 そういうこともあり、動的なアプリケーションからそのランタイム環境にいたるまで、あらゆる要素に関する研究の専門技術を開発しています。 また、もう 1 つの理由は、ある分野で専門技術を開発する唯一の方法は、可能な限りすべての間違いを実際に経験することだからです」

JAVA.NET アンケート

JAVA EE 7 でもっとも重要な改善点は何ですか。

最近の Java.net アンケートで、Java EE 7 のもっとも重要な改善点として Java 開発者コミュニティの圧倒的な支持を得たのは HTML5 のサポートです。 2 週間で合計 1,631 票が投じられ、その最終調査結果を下記に示します。



90% HTML5 のサポート（WebSocket（JSR 356）と JSON 処理（JSR 353）を含む）

3% JavaServer Faces 2.2（JSR 344）

2% JAX-RS 2.0（Java API for RESTful Web Services、JSR 339）

2% Expression Language 3.0（JSR 341）

1% Java Message Service 2.0（JSR 343）

2% その他

Mobile World Congress: モバイルの再定義

Mobile World Congress は携帯電話とモバイル・ネットワーク業界だけを対象としていました。 スペインのバルセロナで 2 月 25 ～ 28 日に開催された今年の会議を見る限り、モバイルの定義が見直され、今や、車、スクーター、建物、人など、あちこちで使用されているセンサーがその対象となっているのは明らかです。マシン・ツー・マシンや Internet of Things という改革がここにはあり、携帯電話はインテリジェントなつながれた世界をつくる多数のコンポーネントの 1 つに過ぎません。 モバイル業界の焦点が音声からデータへと移った今、開発者にとっては世界中にアプリの可能性が散らばっています。

このチャンスを生かすには、決断が必要です。 現在の開発者は、幅広い選択肢から使用するデバイスとその制御方法を選ぶことができ、選択したデバイスおよびプラットフォームが実装を成功させるための鍵となります。 Java はスケーラビリティとサポート性を備えたインテリジェントなデバイスを実現します。 デバイスをリモートから更新することも、ヘッドレスにすることも可能であり、リモート・センサーを使用して、老人が倒れたら病院に連絡し、救急隊員が家のドアを開けられるようにすることもできるのです。

Javaを使用したリモート・センサーによって、倒れた老人の命を救うこともできます。

Javaを使用することでmHealthの立ち上げが容易になります。

JUG による ADOPT-A-JSR FOR JAVA EE 7 への取り組み

Java EE 7 のリリースへ向けた最後の一押しを、世界中の Java ユーザー・グループ（JUG）が支援しました。 Java エンタープライズの開発者によって、Java EE 7 関連の JSR 専用に Adopt-a-JSR for Java EE 7 イニシアチブが開始されました。

参加する JUG は、はじめに、メンバーがサポートを希望する Java EE 7 の JSR を選びます。次に、JUG が Adopt-a-JSR プログラムに加わり、選択した JSR の仕様をレビューし、仕様リードまたは Expert Group に連絡を取る必要があります。 JUG が利用できる JSR サポート・タスクには、リファレンス実装（RI）ビルドの構築とテスト、バグなどの問題の報告、メーリング・リストの管理支援、Technology Compatibility Kit の構築支援があります。

今号の公開時点で Adopt-a-JSR for Java EE 7 イニシアチブに参加しているのは、5 大陸にまたがる 19 の JUG（BeJUG、Campinas JUG、CeJUG、JUG Chennai、Cologne JUG、Congo JUG、FasoJUG、Houston JUG、Hyderabad JUG、Indonesia JUG、Jozi JUG、London Java Community、Madrid JUG、Mbale JUG、Morocco JUG、Peru JUG、Silicon Valley JUG、SouJava、Toronto JUG）です。

Red Hat事業所があるイギリス北東部のニューカッスル大学構内で、クリーン・ルームの冷却および空気ろ過システムを見渡す、Red Hatのエンジニアリング部門担当バイス・プレジデント、Mark Little氏。

JCP Executiveシリーズ

Mark Littleに聞く

Red Hatのエンジニアリング部門担当バイス・プレジデント、Java EE 7およびJCPを語る

STEVE MELOAN

Java Community Process（JCP）から選ばれた Executive Committee のメンバーにインタビューするシリーズの続編です。今回は、Red Hat のエンジニアリング部門担当バイス・プレジデントである Mark Little 氏にお話をうかがいます。当初、Enterprise Linux OS で知られていた Red Hat は、2006 年に JBoss を買収し、そのテクノロジー・ポートフォリオにエンタープライズ・ミドルウェアを加えました。

写真：JOHN BLYTHE

同僚と話し合うLittle氏。

Red Hat は、Java EE 6 と、リリースされたばかりの Java EE 7 の両方の仕様開発に深く関与しました。

Java Magazine：Red Hat は Java EE 7 の発展に大きく関わっています。
CDI（Contexts and Dependency Injection）1.1 および Bean Validation の仕様をリードし、多数の JSR の開発に参加した経験についてお聞かせいただけますか。

Little 氏：当社は Java EE 6 で CDI と Bean Validation の更新でリーダを務めていたこともあり、今回もその指揮を執ることになりました。
また、ご指摘のとおり、JTA（Java Transaction API）、JCA（Java EE Connector Architecture）、JMS（Java Message Service）の更新など、多数の JSR に対して非常に活発に活動しています。

私たちは、Java EE 7 で更新されたすべての JSR に対して何らかの形で積極的に関与するよう努めました。たとえその先導役を担っていなくても、Java EE の採用方法に対して Red Hat のお客様や幅広いオープンソース・コミュニティから見た視点を持ち込むことは重要なことだからです。

Java Magazine：Java EE 7 に含まれる JSR の重要性について、また JSR の改善と発展に尽力された経験について、ご意見をお聞かせください。

Little 氏：何年にもわたって見てきた中で、Java EE アーキテクチャ全体に関わるもっとも重要な追加は、おそらく CDI ではないでしょうか。これは単に当社がリーダーを務めたから言っているわけではありません。本当にグループ全体の総力の結果です。CDI だと言った理由は、標準の Java EE スタックを使用したアプリケーション開発は時として骨が折れると多くのベンダーから聞いていたからです。CDI は開発の大幅な簡素化に貢献しています。つまり、アノテーションは Java 言語にとって非常に重要で、現在では、さまざまな領域で利用されています。

Java EE 6 のリリース以降、CDI に対する機運の高まりが見られました。はじめは Java EE 6 にあまり関心を抱いていなかったものの、今ではエンタープライズ・アプリケーション開発の簡素化手段とみなしている人々が大勢います。そのように見方を変えた人々が何度も口にするのは CDI のことです。当社にとって Java EE 7 で CDI の更新をリードしたいと考えたのはごく当然のことでした。それは、後に Java EE 6 となったバージョンでは実現できなかったことや、Java EE 6 が最終決定された際にユーザーから寄せられたフィードバックの中に、取り入れたいことがあったからです。

プロセスに関して言えば、Java EE 6 のときとほぼ同じ方法を取りました。すべてのプロセスが公開され、すべての参加者が他の参加者が議論している内容を見ることができるようにオープン・メーリング・リストが使用されたほか、問題点の追跡も公開されました。ドラフトは非常に広範な改訂プロセスを通過しています。

Java Magazine：Java EE 6 の Web Profile は Red Hat テクノロジーにどのような影響を与えましたか。また、WebSocket や JSON-P などの新規 API の観点から見て、Java EE 7 でこの影響はどう変化しますか。

Little 氏：スタックの増加や複雑さは Java EE スタックに限ったことではありません。CORBA や DCE などのその他の標準を見ても同じような現象が見られます。Java EE 6 ではこの事実を認識して、Web Profile という概念が導入されました。Web Profile は基本的に、フル・プロファイルの簡易版です。

Web Profile が当社に与えた影響について言えば、当社ではしばらく前から、私たちのコミュニティに対して独自バージョンのプロファイルを提供してきました。提示できる標準はありませんでしたが、スタックを合理化する機能は提供可能でした。たとえば、顧客が Web サービスを望まなければ、当社で提供することができました。

ですから、Web Profile が標準として採用されたことは非常によかったと考えています。これまでに寄せられたフィードバックは至って前向きなものです。Java EE 6 を検討の対象としていなかった人々が、今では Web Profile に目を向け、最終的にフル・プロファイルへアップグレードしています。それは、求める仕様の中に Web Profile には含まれていないものがあるためです。つまり、Web Profile は、より多くのユーザーの利用を促す手段として、また企業側から見て新規顧客を獲得する手段として、非常に優れています。

JAX-RS や CDI などの Java EE 6 に含まれた数多くの新規 API が開発者にもたらすメリットには、大きな注目が集まりました。
Java EE 7 でも JSON-P や WebSocket などの追加によって API が拡張されていますが、このどちらとも、エンタープライズ Java スタックがクラウドやモバイルなどの流れの最前線に立ち続けるためには不可欠な要素です。

Java Magazine：当初は Java EE 7 を対象とし

ていた各種のクラウド関連機能が Java EE 8 に延期されています。この変更についてのご意見をうかがわせてください。また、現在のクラウド環境ではどのように Java EE が利用されており、Java EE 7 以降のリリースではどのようにテクノロジーが発展していくとお考えですか。

Little 氏：コミュニティや顧客との話合いから分かったのは、実装を変更することなく、自社のインフラストラクチャからどこか別のインフラストラクチャへとアプリケーションを移行できる機能が求められているということです。 そして、この要望を実現するための明白な方法は、それまで独自のハードウェアを使用して実行していたプラットフォームをクラウド上に確保することです。 非常に早い段階から、Red Hat は当社の実装が Infrastructure as a Service 上で稼働し、ひいては Platform as a Service 上で稼働するための取組みを行ってきました。

Java EE 6 の登場によって、クラウド上のプラットフォームを実現するための近道が提供されました。Java EE 6 に導入されたプロファイルを使用すると、アプリケーションをクラウドで実行したいと考える顧客に対して、標準の Web Profile と標準のフル・プロファイル・プラットフォームを簡単に提供できるようになったためです。 当社は独自の Platform as a Service を 2011 年に発表しましたが、これは当初、Java EE 6 に準拠した Red Hat アプリケーション・サーバーの事前リリースをベースにしたものでした。

次にリリースしたフル実装の EAP6 では、当社の Platform as a Service 製品である OpenShift を公開しました。

重要な仕様

「何年にもわたって見てきた中で、Java EE アーキテクチャ全体に関わるもっとも重要な追加は、おそらく CDI ではないでしょうか」

当社が Java EE 7 に取り組んでいた当初は、Java EE のクラウド対応機能を向上させることを目的とした新規 JSR が相当数ありましたが、先程言ったとおり、Java EE 6 は現在のクラウドで極めて有効に利用されています。モジュール化とマルチテナントの点から見て、改善が見込める領域もいくらかありますが、

これらの機能が延期されたことには賛成です。そのおかげで Java EE 7 をスケジュールどおりにリリースできましたから。 これらの機能は次のリリースに採用される予定ですから、Java EE 8 は進化するクラウドに一層適したものになるでしょう。

Java Magazine：JBoss Developer Studio IDE では、Java EE 7 の新機能をどのように反映し、利用する予定ですか。

Little 氏：私たちが目指しているのは、JDBS が常に実際の開発者に機能を提示する先駆けとなることです。そうすることで、どこに問題があるか素早く特定できます。 たとえば、CDI の変更が適切でなかった場合、JDBS を通じて開発者から多数のフィードバックが寄せられるでしょう。 したがって、できるだけ早急にこれらの機能を IDE に取り込み、人々が簡単なチェックを実施できるようにすることが重要なのです。

Java Magazine：Java EE 7 では多くの古い機能（JSR 77、
JSR 88、JSR 93、JSR 101 など）が削除されましたが、Red Hat の Java EE 7 製品ではどのように対応する予定ですか。

Little 氏：JSR が削除されたのは今回が初めてではありません。 今回削除された JSR への対応は通常どおり、Java EE 7 の仕様に含まれなくなったものは Java EE 7 に準拠した実装から削除するというものです。

しかし、当社には Java EE 6 を実装した製品として、コミュニティ・バージョンの AS7（JBoss Application Server 7）と製品バージョンの EAP6（JBoss Enterprise Application 6）があります。

EAP6 には 7 年間のサポートがあり、これらの機能が本当に必要な顧客は、その後 5 ～ 6 年間のサポートを受けることができます。 オプションになった JSR は、通常、何らかの形でサポートを続けており、Java EE スタックから正式に削除されるまでは削除しません。

Java Magazine：JCP への関わりを大局的に見た場合、その他のメンバー企業と協業する際に生じる関心事の違いをどのように解決、調整していますか。

Little 氏：Java および Java EE が IT 業界の大部分にとって非常に重要であることは誰もが理解しています。 したがって、私たちには

Little氏によると、オープンソース・コミュニティが“Red Hatで実施されるすべてのことを左右します”

イノベーションの役割

「Executive Committeeで現在試みていることの1つは、Java EE 8やJava EE 9への採用が検討されている分野において、上流のオープンソース・コミュニティでのイノベーションを促すことです」

Java EEを発展させるべく協力する義務があります。これはOASISやW3C、またはその他の標準グループに参加する際となんら変わりはありません。問題が生じる場合もあるでしょうし、要望が必ずしもすべて通るわけでもありませんが、互いに歩み寄る必要があるのです。

Java Magazine：現行標準の維持と将来的なイノベーションの推進のバランスを取る際、どのような問題が発生しますか。

Little 氏：非常に危険なのは、時期尚早な標準化です。優れた標準はすべて、経験に基づくものであり、標準化が早すぎると、結局は目的に合わないAPIやデータ形式が作られ、誰も使用しなくなります。

しかし、これは一種のトレードオフであり、標準化が遅すぎるとベンダーによる囲い込みにつながったり、同じことを実行するために複数の異なる方法が存在したりする結果になります。Executive Committeeで現在試みていることの1つは、Java EE 8やJava EE 9への採用が検討されている分野において、上流のオープンソース・コミュニティでのイノベーションを促すことです。

Java Magazine：どうすればJCPはもっとJavaコミュニティの役に立つことができると思いますか。また、より積極的な参加を促す方法にはどのようなものがありますか。

Little 氏：Adopt-a-JSRなどの、さまざまなJavaユーザー・グループがこれまでに行ってきた作業は素晴らしいものです。当社で実施することはすべてオープンソース・コミュニティによって左右されています。各種のJSRで提案しようとしている事項に関して、実際に文書化するよりもずっと前に多数のフィードバックが寄せられるのです。私たちはこのような方法で、私たちのコミュニティをExecutive Committeeに引き入れる努力をしています。

Java Magazine：昨年のJavaOneでCameron Purdyとともに開催したセッションでは、JVM（Java仮想マシン）上で稼働するJava以外の言語について考察されましたね。たとえば、Ruby開発者がJava EEスタックを活用して、JMSやEJB（Enterprise JavaBeans）などの機能を利用するには、どのような方法がありますか。この傾向について、Javaのコミュニティやエコシステムの拡大という観点からお話し願えますか。

Little 氏：ここ数年で、JVM上で稼働するJRuby、Clojure、Scalaなどの言語の採用が増えてきています。当社ではCeylonという独自言語を開発しているところです。このようなコミュニティと協力して試みていることは、開発者が、高パフォーマンスのメッセージングやトランザクション、またはデータベースを必要とする場合、一から開発をやり直すのではなく、Java EEスタックという観点で完璧なソリューション・セットがすでにある場合はそれを利用するよう働きかけることです。

そこで私たちは、開発者がメッセージングなどの特定のAPIに何を期待しているのかを明らかにし、幅広い開発者コミュニティとともにコードを作成し、その下に当社のスタックまたは少なくともその一部を重ねる取組みを行っています。つまり、Ruby開発者はAPIを定義し、私たちは当社のJMS実装に対するこのAPIのバインディングを提供するのです。

新しい言語コミュニティにもっとも避けてほしいのは、20年前と同じバグを再び作り出すことです。すでに存在するものを土台として利用し、これを改善する方がずっと効率的です。実際に、TorqueBoxというRubyプロジェクトなど、いくつかのプロジェクトは大きな成功を収めています。最終的に、開発者はアプリケーション・サーバー上で処理を実行していることや、内部でJava EEスタックを利用していることを知る必要さえもなくなります。 </article>

Steve Meloan：元C/UNIXソフトウェア開発者。Wired、Rolling Stone、Playboy、SF Weekly、San Francisco Examinerなどの各誌にWebやインターネットに関する記事を執筆している。近著はサイエンス・アドベンチャー小説の『The Shroud』。また、The Huffington Postに定期的に寄稿している。

LEARN MORE

- Java Community Process
- Red Hat

オフィスの近くでボールをパスするESPN Technologyのエンジニアリング担当シニア・ディレクターManny Pelarinos氏（左）と、エンジニアリング担当ディレクターSean Comerford氏。

写真：STAN GODLEWSKI/GETTY IMAGES

スラムダンク

Java、**ESPN.com**を勝者に **DAVID BAUM**

企業概要

ESPN, INC.

espn.go.com

ESPN.comはESPNの公式Webサイトかつ ESPN, Inc.の事業部門

本社所在地：
コネチカット州ブリストル

業界：
メディア、ニュース、出版

売上：
82億ドル

従業員数：
6,500名（全世界）

使用しているJavaテクノロジー：
Java SE 1.6および1.7、Java EE 6

ESPN本社にあるスポーツの殿堂に立つPelarinos氏とComerford氏。

Kobe Bryant が真髄を見せたこの瞬間を、ロサンゼルス・レイカーズのファンは決して忘れないでしょう。 対マイアミ・ヒート戦 ESPN ナイト・ゲームの土壇場で、最後のプレイとなるボールの権利を得たレイカーズは 2 点差を追っていました。 Ron Artest が Bryant に素早くパスを送りましたが、Dwyane Wade の果敢なディフェンスによってシュート・チャンスは阻まれました。 残り時間がほとんどゼロになった瞬間、右にフェイントをかけて左にかわした Bryant はついに、大きく広げた Wade の腕越しに 3 ポイントのロング・ジャンプ・シュートを放ちました。 ボールがバックボードに当たってリングを通過した瞬間、ブザーが鳴り、マイアミ・ヒートに対するロサンゼルス・レイカーズの 8 連勝が決定しました。 スタジアムは湧き上がり、至るところからレイカーズ・ファンの勝利の歓声が聞こえてきました。 ESPN Technology で働くエンジニアの努力のかいあって、何百万もの人々が無料のモバイル・アプリを通じて、スマートフォンやラップトップ、タブレット上でほぼ瞬時に最新情報を得られました。

インターネットでのスポーツ情報提供大手である ESPN.com は、今回のような生中継イベントのライブ・スコアや、最新ニュース、試合統計データ、スケジュール、選手の最新情報、ファンタジー・ゲーム、ビデオを WatchESPN を通じてファンに届けています。 National Basketball Association(NBA) のゲームは、ESPN が取り扱う何千ものイベントの 1 つです。 何百万ものスポーツ・ファンが、フットボールや野球、ホッケー、サッカー、NASCAR レース、他にも多くの種類の試合や競技会、トーナメント、対戦の情報を追ってこのサイトにアクセスしています。FIFA ワールドカップ・サッカー選手権などの大規模なスポーツ・イベントの期間中には、ESPN の処理量は 1 秒あたり 10 万ヒットを上回ります。 ESPN の利用者向け Web サイトは、Java の力を借りてこの圧倒的な負荷に対応するよう設計されています。

「私たちが構築し、増強しているシステムは、何百万にも上るファンのニーズを満たすスケーラビリティを備えている必要があります」と、ESPN が使用するテクノロジーの構築、設計、調査、実装、保守を担う部門である ESPN Technology のエンジニアリング担当シニア・ディレクターである Manny Pelarinos 氏は言います。「エンジニアリングの観点から見てもっとも重要なタスクの 1 つは、並はずれたパフォーマンスを発揮し、水平方向のスケーラビリティを備えたシステムの構築です。 Java は常に私たちの DNA に引き継がれており、立上げ時からずっと、サイトの構築には Java を使用しています。 それどころか、ESPN はサーブレット仕様が作られる前から Java を使用しているのです」

ESPN.com をクリックしてアクションを見てみると、ひいきのチームのスコアボードやゲームのハイライト、全選手のプロフィールが表示されます。 最新ニュースを読んだり、最近のスポーツ・イベントのリプレイを視聴したり、ポーカー・トーナメントからテニスの試合まで、すべてを知ることができます。

数ページのシンプルな Web ページに見えるものが、実は、数百台のサーバーに散在する何十ものデータベースやアプリケーションで構成された、大規模デジタル・インフラストラクチャのフロントエンドになっています。 Pelarinos 氏と彼のチームは

瞬時アクセス

ESPN Technology のエンジニアは、無料のモバイル・アプリを通じて何百万ものファンに、Kobe Bryant が真髄を見せた瞬間をほぼ瞬時にレポートしました。

JAVA EE 7 について

「Java EE 7 がリリースされ、JMS での使いやすさが改善されたことによって、将来的にコードを簡素化し、この重要なアーキテクチャ領域に関する情報をより素早く新しい開発者に提供できるようになるでしょう」

サーブレットを使用して動的に Web ページを生成し、静的な Web マークアップ・コンテンツに変換しています。

生成されたページは Java VM の実行環境を介して、ESPN のサーバ群に集約されて実行されます。

確立された地位

基幹となるこれらの Java テクノロジーは、スポーツ関連の見出し、ニュース記事のほか、サイトでのコンテンツのレイアウト方法を示す構成データなどの重要情報を保持するコンテンツ管理システムに加え、1997 年に開設された ESPN の人気 Web サイトの基盤となっています。 この機敏な Java インフラストラクチャによって、試合結果、ゲームのハイライト、通算成績に加え最新のニュースが ESPN のモバイル・エクスペリエンスおよび Web サイトに提供されています。 情報は、スポーツ、チーム、選手、コラムニスト別に分類され、ユーザーの好みに合わせてパーソナライズされたコンテンツをその分類した情報に対応づけます。

「コンテンツ管理システムは当社の Web サイトの原動力です。 編集者から送られたすべてのテキスト・データはこのプラットフォームに格納されています」と、Pelarinos 氏は言います。

続けて Pelarinos 氏は次のように説明しています。「当社は NBA との契約により、NBA のシステムから直接データを利用しています。データはフィードの形をとり、Java Message Service（JMS）キューに挿入されます。 これらのフィードは、Message-Driven Bean によって Java ドメイン・オブジェクト・モデルにマーシャリングされます」

Java SE および Java EE アプリケーションでリレーショナル・データを管理するためのフレームワークである Java Persistence API（JPA）を使用して、データはデジタル消費向けに最適化された別の ESPN データ・ストアに永続化されます。 そのすぐ後に、ESPN アプリケーション層に含まれるすべてのキャッシュが更新され、何百万もの熱心なスポーツ・ファンに最新情報が送信されます。 すべてのエンティティはほぼリアルタイムで更新されており、毎秒 1 ～ 2 回、別のシステムが特殊な API をポーリングして更新情報をファンに届けています。

「スコアはほぼ即座に更新され、ファンに表示される時計は実際の試合の時計をリアルタイムで表しています」と、Pelarinos 氏は言います。「データ更新があまりに速いので、まるでスタジアムにいるかのように感じられるんです」

技術の MVP

このようなレベルの速度、スケーラビリティ、信頼性を実現するため、ESPN.com はキャッシング・アーキテクチャと JPA を利用しています。

「JPA オブジェクト・モデルを利用することで、Web アプリケーションやモバイル・サービスを利用したアプリへのデータ送信が可能になります」と、ESPN Technology のエンジニアリング担当ディレクターである Sean Comerford 氏は説明します。

もう 1 つの重要なアーキテクチャ要素は、メッセージ指向のミドルウェア API である JMS であり、
ESPN.com が持つ多数のサーバーおよびアプリケーション間で情報を送受信します。 JMS は、Java アプリケーションによるメッセージの作成、送信、受信、および読取りの方法を制御し、
分散アプリケーションの別々のコンポーネント間で、疎結合された信頼できる非同期通信を実現します。 このような特性は、ユーザー・ベースが絶え間なく拡大を続けるなか、安定した通信を維持する必要のあるサイトにとって不可欠です。 加入者は、更新情報をクリックすることで、該当するイベントの追加情報を確認したり、ESPN の多彩なコンテンツ・データベースにドリルダウンしたりすることができます。 また、スポーツ情報を都市別にソートして、ローカル・ニュースを読んだり、近く予定されている地元のイベントや試合、対戦を見つけたりすることができます。

Comerford 氏は次のように述べています。「私たちは Java メッセージングを大いに利用しています。過去 5 年間、私たちは非同期メッセージングの標準インタフェースとして JMS を採用してきました。 JMS の働きは信頼できるものであり、メッセージを重複なしで確実に処理するための多くの課題を解決してくれます」

JMS のもう 1 つの大きな利点は、パブリッシュ・サブスクライブ・モデルです。ESPN はこのモデルを利用することで、内部システムと情報を共有し、パブリッシャを変更することなく新規システムを追加できるようになりました。 スポーツ、チーム、選手に関するコンテンツは、ニュース記事や最新情報とともに ESPN の JMS ハブを通過し、そのコンテンツを見ることができる多大な数の利用者によって消費されます。 また、開発や QA などの他の環境にコンテンツをリアルタイムで複製する役割を果たしているのも同じ JMS ハブです。

「このアプローチはまさにサービス指向であり、コンテンツをパブリッシュするアプリケーションと消費するアプリケーションはきれいに切り離されています」と、Comerford 氏は続けます。「実際、Java EE 7 がリリースされ、JMS での使いやすさが改善されたことによって、将来的にコードを簡素化し、この重要なアーキテクチャ領域に関する情報をより素早く新しい開発者に提供できるようになるでしょう」

JPA を通じて、ESPN.com はデータベース層でも同じレベルの抽象化を実現しています。「JPA コードは読みやすく、分かりやすく、保守も簡単です」と、Comerford 氏は言います。「このアーキテクチャによるもう 1 つのメリットはプラットフォームに依存しない点であり、必要があれば極めて容易に基盤データ・ストアを変更できます」

将来へのファスト・ブレイク

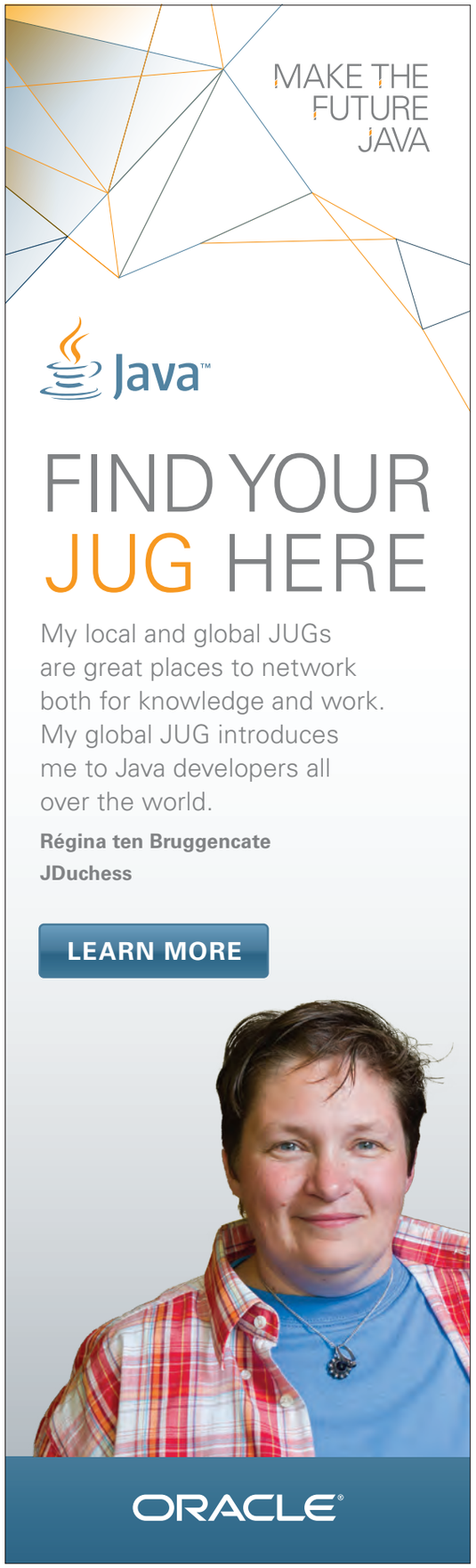
世界でもっともアクセス数の多い Web サイトの 1 つを改良し続けている ESPN Technology は、イノベーションの最先端を走っています。

「Java のパフォーマンスと信頼性を利用することで、私たちは極めてスケーラビリティの高いシステムを提供し続けることができます」と、Pelarinos 氏は言います。「そもそもデジタル・メディア自体が最先端を走る存在ですが、ESPN.com では常に新しいアイデアとテクノロジーを試しているため、いつでも新鮮で魅力的な仕事に取り組めます」

「結局のところ、私たちが目指しているのはスポーツ・ファンの期待に応えることなのです」と、Pelarinos 氏は結びます。「この目標を確実に実現するため、今後も新技術に投資していくつもりです」

</article>

David Baum：カリフォルニア州サンタバーバラを拠点とし、革新的なビジネス、最新テクノロジー、魅力的なライフスタイルについて執筆活動中。



銀行業務における Java

Java EE ベースの新システム、ブラジルの複雑な債権譲渡取引システムに安定性と透明性をもたらす

PHILIP J. GILL

サンパウロ（ブラジル）にある TIVIT 本社の外を歩く Cláudio Silveira 氏（左から）、Einar Saukas 氏、Fabiano Marques 氏。

写真：PAULO FRIDMAN

TIVITオフィスで作業するMarques氏、Saukas氏、Silveira氏

企業概要

TIVIT
tivit.com.br

本社所在地：
サンパウロ（ブラジル）

業界：
ITサービス

従業員数：
28,000

売上：
15億レアル（2012年概算）

使用しているJavaテクノロジー：
Java EE 6

ブラジル経済は世界でもっとも規模が大きくダイナミックな経済の1つですが、2010年、世界的な景気後退ではなく、ブラジル中央銀行Banco Central do Brasilの驚くべき発見によって、10年に及ぶブラジルのサクセス・ストーリーが危機にさらされました。いくつかの銀行が債権譲渡、つまり消費者取引、融資取引、リース取引によって生じる貸付契約を重複して転売していたのです。ブラジル中央銀行によって判明したこの二重取引によって、ブラジルの金融制度に42億レアルの損失が生じており、国家レベルの金融危機に陥る可能性がありました。

「これらの銀行の活動は、明らかに違法であることに加え、ブラジルに金融システムへの不安を生み出していました」と、サンパウロを本拠とし、Javaを専門とする企業Summa Technologies do Brasilのプリンシパル・アーキテクトであり、同様にサンパウロに本拠を置くITサービス企業TIVITのコンサルタントであるEinar Saukas氏は述べます。「銀行が債権購入能力とその補償能力における信用を失ったことで、高まったリスクを埋め合わせるために金利が上昇しました」と、Saukas氏は説明します。

Saukas氏は続けます。「取引高が減少したことで全般的な経済成長が低迷し、何百万もの消費者や企業に対する金利が上昇しました」

この不透明感を和らげ、ブラジル経済に安定を取り戻すため、ブラジルにおいて米国連邦準備銀行と同様の役割を果たすブラジル中央銀行は、このような二重取引によってブラジルの金融システムおよび経済が二度と脅かされることのないようにするための取組みに着手しました。

この取組みが、Central de Cessão de Crédito（C3）、英語ではCentral Credit Assignment（集中債権譲渡）です。C3はスケーラブルで高パフォーマンスの分散取引システムであり、現在、すべての債券譲渡取引の整合性を監視しています。Javaで構築されたC3は1年未満で導入され、ブラジルの金融システムはいち早く安定を取り戻しました。さらに、「リスクが緩和され、信用が回復し、金利が下がり、経済成長が再び始まりました」と、Saukas氏は加えます。C3はまた、初代のDuke's Choice Award Latin Americaを受賞したプロジェクトの1つとなりました。別の2つのプロジェクトでDuke's Choice Awardを獲得した経験を持つSaukas氏にとっても、この賞は格別な名誉でした。

C3の構築に際してブラジル中央銀行はTIVITと契約を結び、TIVITはJava EEでシステムを構築することを決めました。「Java EEプラットフォームはC3にとって当然の選択でした」と説明するのは、TIVITのアプリケーション運用ディレクターであるCláudio Silveira氏です。「ブラジルではJavaが銀行取引アプリケーションの標準になっています」

消費者文化

ブラジルの天然資源、農産物、加工品、ビジネス・サービスを求める世界的な需要によって、1999年以降のGDP成長率はほぼ毎年5%を超え、2011年、南米最大のこの国はイギリスをしのいで世界第6位の経済大国になりました。

また、この成長によって3,100万人のブラジル人が中産階級へと押し上げられ、このクラスは今や、1億9,300万人のブラジル国民の半数近くを占めるまでになりました。 他の好調な経済圏と同様に、中産階級の成長は自動車用消費者ローン、住宅ローン、リースに対する需要によって活気づけられ、好景気をもたらしました。 現在、ブラジル国民のおよそ3分の1が何らかの負債を抱えています。

新たな貸付に備える財源としてまとまった現金を確保するため、ブラジルの銀行は、米国やその他各国の銀行と同様に、他の銀行や金融機関に長期投資として貸付契約を転売します。「たとえば、新車を購入するためにローンを組もうとして銀行を訪れた顧客は、融資を得るための契約書に署名します」と、C3開発に関してTIVITと相談したSaukas氏は説明しま

す。「これらの機関は融資に特化しているため、新しくローンを組む別の顧客のために、より多くの資金を迅速に確保する必要があります。こうして、融資契約が別の銀行に転売されることになります」

ブラジル中央銀行の監督下で運用するC3の第1段階では、関係する銀行間でのすべての債権譲渡取引を監視し、銀行から別の金融機関へと債権が転売されるたびに記録を残すことが目的とされました。「債権譲渡の交渉をC3に登録する必要があるため、同じ債権が別々の2つの銀行に転売されそうになると、すぐに検出されます」と、Saukas氏は言います。

2012年に導入された第2段階では、別の機能が追加され、CETIPと呼ばれるブラジル中央銀行の決済システムを介した、実際の転売の取決めと管理が可能になりました。「購入者と販売者はC3を介して債権の転売を取り決めるため、購入者はすべての情報を反対側から確認することができます」と、Saukas氏は続けます。「債権の所有権はC3システムが管理しますが、実のところ、債権に対する支払いはCETIPとブラジル中央銀行が管理しています。 C3はある銀行から別の銀行へと債権の所有権を移します。現在、システムは完全に自動化されています」

パイプライン・アーキテクチャ

スケーラブルでパフォーマンスが高く、セキュリティと信頼性にも優れたトランザクション指向アプリケーション向けのJavaプラットフォームとしてJava EEを選んだTIVITの選択は、単純明快なものでした。 Java EEは、ブラジルの銀行業界に広く普及していることに加えて、Java Message Service（JMS）やJava Open Transaction Manager（JOTM）などの、複雑なC3システムに必要なAPIセットを包括的に統合した唯一のプラットフォームでした。

さらに、Java EEはオープンソース・コミュニティによって幅広くサポートされています。C3の開発に使用されたオープンソース・テクノロジーおよび開発ツールには、XA分散トランザクション・プロトコル、Spring Framework、JBoss Serialization、Apache Commons、dom4j、XStream、Hudson、Sonar、Nexus、JUnit、Mockito、Maven、およびEclipseが含まれています。

Java EEが選ばれたもう1つの理由は、同様の大量処理要件を持つシステムを、以前にTIVITがJava EEを使用して開発した経験があったためです。Débito Direto Autorizado（DDA、直接承認支払）と呼ばれるこのシステムは、リアルタイムの電子請求/決済システムです。C3と同様にJavaで構築されたDDAは、ブラジルの参加銀行間で決済データを交換することで、ブラジル国民が電子的に請求書を受け取り、支払いを実行できるようにするものです。 2009年10月に稼働を開始したこのシステムには、現在、860万人を超える銀行顧客と7億2,900万件を上回る請求書が登録されており、ブラジル全体の決済額の10%を処理しています。

TIVITはDDAの基盤ソフトウェア・インフラストラクチャを再利用できたため、C3システムの開発から1年たらずで本番稼働に至りました。「DDAと同じソフトウェア・インフラストラクチャの下層を使用することで、開発労力が大幅に削減されました」と、TIVITのチーフJavaアーキテクトであるFabiano Marques氏は言います。「このため、初期構想から本番までのC3プロジェクト全体が1年以内で完了し、C3の初期フェーズでの開発作業の大半は2か月かかりませんでした」

近代の象徴

ブラジルの連邦首都ブラジリアは、近代建築を象徴する傑作として知られています。

Saukas氏とSilveira氏がプロジェクトについて話す間にプレゼンテーションを準備するMarques氏。

ブラジルにおけるJava

ブラジルは世界最大級の経済だけでなく世界最大級のJava コミュニティを誇っており、Java.net によると 14 万 2,000 名を超える Java 開発者を抱えています。

実際、ブラジルは世界最大と第 2 位の Java ユーザー・グループ（JUG）の本拠地であり、それぞれの JUG には 4 万名を超えるメンバーがいます。 世界最大の JUG である SouJava は、ブラジル最大の都市であり、国際的なビジネスおよび金融の中心地であるサンパウロを拠点としています。 2 番目に大きい DFJUG は、ブラジル中央部の高原地帯にある近代主義の連邦首都地区、ブラジリアに拠点を置いています。

サンパウロに本社を持つ Summa Technologies do Brasil のプリンシパル・アーキテクトである Einar Saukas 氏は SouJava の創設者であり、Duke's Choice Award を受賞した 3 つのプロジェクト（Central de Cessão de Crédito（C3）分散トランザクション・システムを含む）に従事した経験を持ちます。Saukas 氏によると、Java の高い支持率の一因はブラジル政府による奨励にあります。「数年前、ブラジル政府は何百万もの文書が Microsoft Word という専有形式で保存されていることに気づきました」と、Saukas 氏は続けます。「政府はベンダーによる囲い込みのリスクを避けるため、国家の技術部門に Java などのオープン標準を採用するよう奨励しました」

Einar Saukas

Saukas 氏は C3 以外にも、ブラジル連邦政府向けの統一医療システム（2003 年）およびサンパウロ市による統合患者スケジューリング・システム（2005 年）という 2 つの主要な Java アプリケーション開発に貢献しており、どちらも Duke's Choice Award を受賞しました。

このソフトウェア・インフラストラクチャの下層には、大量トランザクションと大容量データに対応する目的で特別に設計された独自の " パイプライン " アーキテクチャが備わっており、このアーキテクチャはバックエンドの DB2 データベースと階層型ストレージ管理システムに接続されています。

「ブラジルの複雑な金融システムに必要な柔軟性を獲得するため、C3 システムは、JMS キューを介して接続されたトランザクション型の Enterprise JavaBeans（EJB）として構築されています」と、Saukas 氏は説明します。「リクエストは ' パイプライン ' の役割を果たす一連の EJB によって処理されます。パイプライン内では、それぞれの Bean が内部の JMS キューを読み取り、1 つまたは複数の特定のタスクを実行して、次の Bean に結果を転送します。 この EJB は JOTM を使用することで、XA を介して JMS と DB2 処理間でのトランザクションの一貫性と整合性を保証しています」

「また、JMS キューを介した統合によってロードバランシングとフォルト・トレランスも自動化されます。別々のサーバーにある複数の Bean インスタンスは、各 JMS キューからのリクエストを、利用可能になり次第処理するため、一元化された分散ロジックは必要ありません」と、Saukas は続けます。

現在までのところ、C3 が処理したクレジット分割払いは 7 億件（1,270 億レアル相当）を上回ります。 より重要なことは、C3 はブラジルの金融システムにおける安定性と確実性の回復に貢献したことです。「Brazilian National Banks Federation とブラジル中央銀行はともに、債券譲渡における運用リスクの軽減と金融機関による貸付、融資、リース業務の金利引下げを C3 の功績として認めています」と言う Saukas 氏は、このことが一般的なブラジル国民にも確実に良い影響を与えていることに言及しました。「金利の引下げは、現在ローンを抱えている 6,100 万人のブラジル国民に直接的なメリットをもたらします」**</article>**

経済大国

2011年、ブラジルはイギリスをしのいで世界第6位の経済大国になりました。

Philip J. Gill：カリフォルニア州サンディエゴを活動拠点とするフリーランス・ライター兼編集者。20年もの間Javaの動向を追い続けている

パート1
Java EE 7を360度探索する

Java EE 7に追加された重要な新規APIの使用方法について学ぶ

MAX BONBHEL

BIO

Jean-François "Max" Bonbhel：11年以上の経験を積んだITプロフェッショナルであり、Fujitsu Canadaのシニア・コンサルタント兼トレーナー。Java、Java EE、Microsoft、Oracle、オープンソースなどの各種テクノロジーとプログラミング言語に従事。また、アフリカでのJavaユーザー・グループ拡大の先導役。

本記事は全3回シリーズの第1回です。このシリーズでは、Java EE 7で拡張された機能や、HTML5、WebSocket、JSON処理などの比較的新しいWeb標準を使用して、最新型のエンタープライズ・アプリケーションを構築する方法について説明します。

本記事では、NetBeans IDE 7.3とGlassFish 4を使用して完全な3層エンド・ツー・エンド・アプリケーションを構築する方法を詳しく見ていきます。Java EE 7に搭載されている以下のテクノロジーを利用します。

- Java Persistence API（JPA）2.1（JSR 338）
- JavaServer Faces（JSF）2.2（JSR 344）
- Contexts and Dependency Injection（CDI）1.1（JSR 346）

注：本記事で作成するアプリケーションのすべてのソース・コードは、こちらからダウンロードできます。

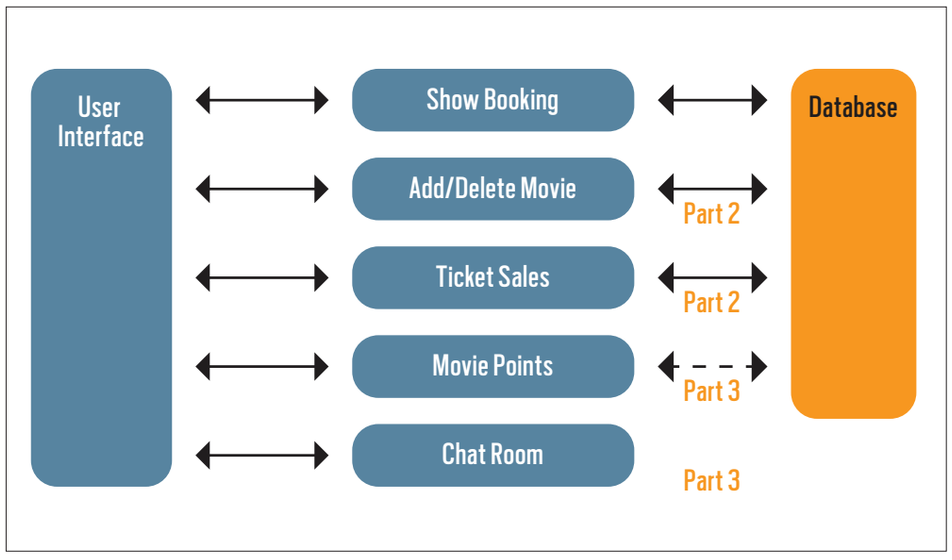


図1

写真：ALLEN MCINNIS/GETTY IMAGES

Java EEとは何か

Java Platform, Enterprise Edition（Java EE）は、エンタープライズ・アプリケーションの開発と実行のための専用のJavaプラットフォームです。スケーラビリティ、信頼性、セキュリティに優れた、大規模かつマルチティアのビジネス向けネットワーク・アプリケーションを作成するために必要となる、あらゆる機能が搭載されています。

前提条件

本記事では、サンプル・アプリケーションを動作させるために十分な説明を行いますが、Java EEの包括的な解説は行いません。本記事は、Enterprise JavaBeans（EJB）、JPA、CDIなど、Java EEの基本概念について知識を有する読者を対象としています。これらの基本概念は、The Java EE 6 Tutorialで学習できます。

本記事で説明するアプリケーションの開発には、次のソフトウェアを使用しました。これらのソフトウェアをダウンロードし、インストールしてください。

- JDK 7
- NetBeans IDE 7.3以降の「すべて」または「Java EE」バージョン
- GlassFish 4.0 b80

注：本記事のテストには、NetBeans IDEの執筆時点での最新バージョン（バージョン7.3）を使用しました。

本記事は、オラクルのJava エバンジェリストであるArun Guptaが作成し公開しているコードに基づいています。

NetBeans IDE 7.3を使用した現実的なエンタープライズ・アプリケーションの作成

この実践セクションでは、現実的なJava EE 7エンタープライズ・アプリケーションを段階的に構築していきます。これにより、Java EE 7で新たに機能強化された重要なAPI（JPA 2.1、JSF 2.2、CDI 1.1

フロー	説明
ユーザー・インタフェース	全体的に JSF で記述
■ 予約画面の表示 ■ 選択された映画およびすべての映画の上映時刻の表示 ■ 選択内容の確定	軽量 EJB BEAN から JPA を使用してデータベースと通信

表1

など）の基礎を簡単に学習できます。

作成するアプリケーションは、典型的な 3 層 Java EE 7 Web アプリケーションであり、7 つのスクリーン（上映室）がある複合型映画館の上映時刻を参照して予約を行うことができます。

図 1 は、本記事および今後のパート 2 とパート 3 で追加し、説明する機能を示しています。

表 1 は、本記事で設計するコンポーネントと、各コンポーネントの実装で使用するテクノロジーの詳細です。

実行するタスクは次のとおりです。

- GlassFish および NetBeans IDE 7.3 のインストールと設定
- サンプルの Maven アプリケーションのロード（手順と説明に従って実施）
- 設定済みのデータソースの確認と、データベースのアーティファクト（表など）の生成
- いずれかのスクリーンでの特定の上映を予約する Web ページの作成
- コードの追加と既存コードの更新（これにより、Java EE 7 プラットフォームにおけるさまざまなテクノロジー・スタックの使用方法を示します）
- サンプル・アプリケーションの実行とテスト

NetBeans IDE 7.3 および GlassFish 4 のインストールと設定

最新バージョンの NetBeans IDE および GlassFish をダウンロードしてインストールし、開発環境を構築します。

1. 最新バージョンの NetBeans IDE のダウンロードとインストール：
   a. NetBeans IDE 7.3 以降の「すべて」または「Java EE」バージョンをダウンロードします。
   b. こちらの手順に従ってインストールします。
2. GlassFish サーバーのダウンロードと NetBeans IDE での設定：GlassFish をアプリケーション・サーバーとして使用するために、NetBeans IDE で明示的に指定します。
   a. GlassFish 4.0 をダウンロードします。
   b. ダウンロードした圧縮ファイルを、glassfish4 フォルダを配置するディレクトリに解凍します。
   c. NetBeans IDE を起動し、「**Services**」タブをクリックします。
   d. 「**Servers**」を右クリックし、「**Add Server**」を選択します（**図 2**）。
   e. **Add Server Instance** ウィザードで「**GlassFish Server 3+**」を選択します（**図 3**）。
   f. **Name** フィールドに GlassFish4.0b80 と入力し、「**Next**」をクリックします。
   g. 「**Browse**」をクリックして、GlassFish ファイルを解凍したディレクトリに移動し、glassfish4 フォルダを選択します。
   h. 「**Finish**」をクリックします。

以上で、開発環境の準備が整いました。

サンプル・アプリケーションの確認

ここでは、サンプル・アプリケーションをダウンロードして 内容を確認し、アプリケーションのアーキテクチャを理解します。

1. 初期状態の NetBeans プロジェクトのダウンロード：
   a. サンプル・アプリケーションをこち

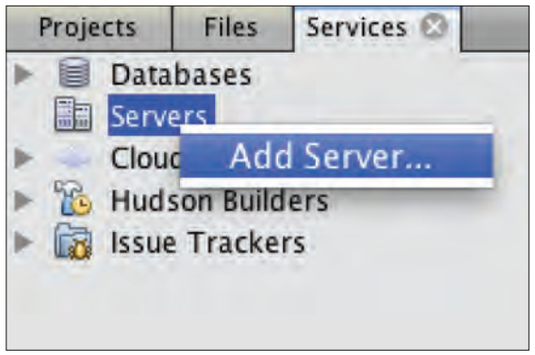


図2

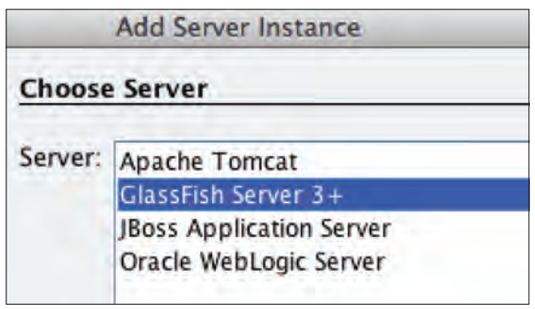


図3

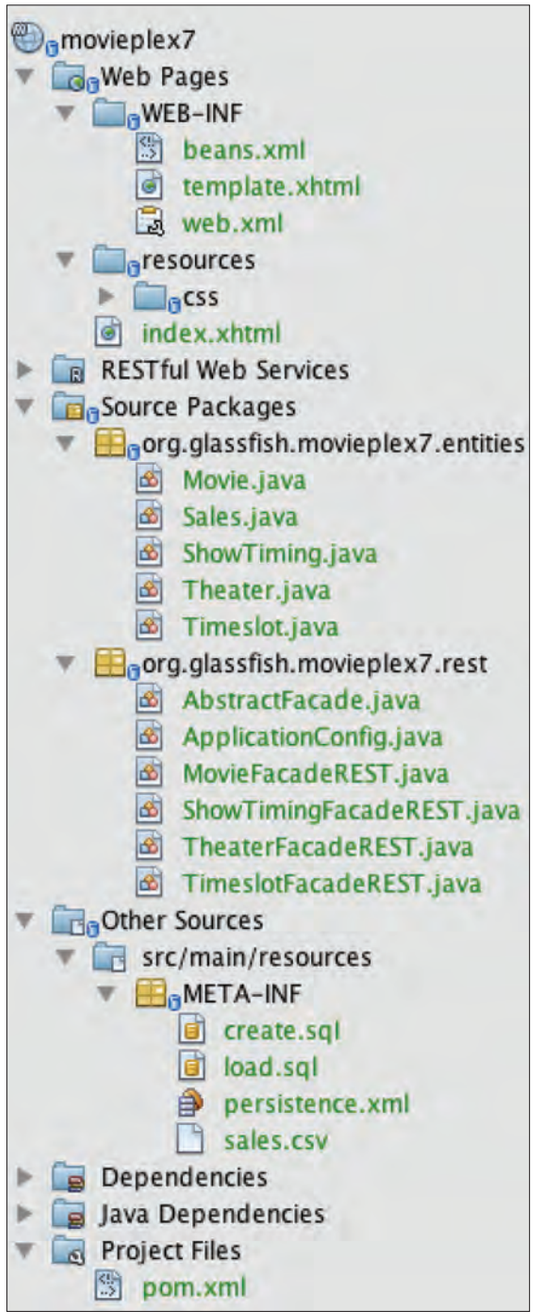


図4

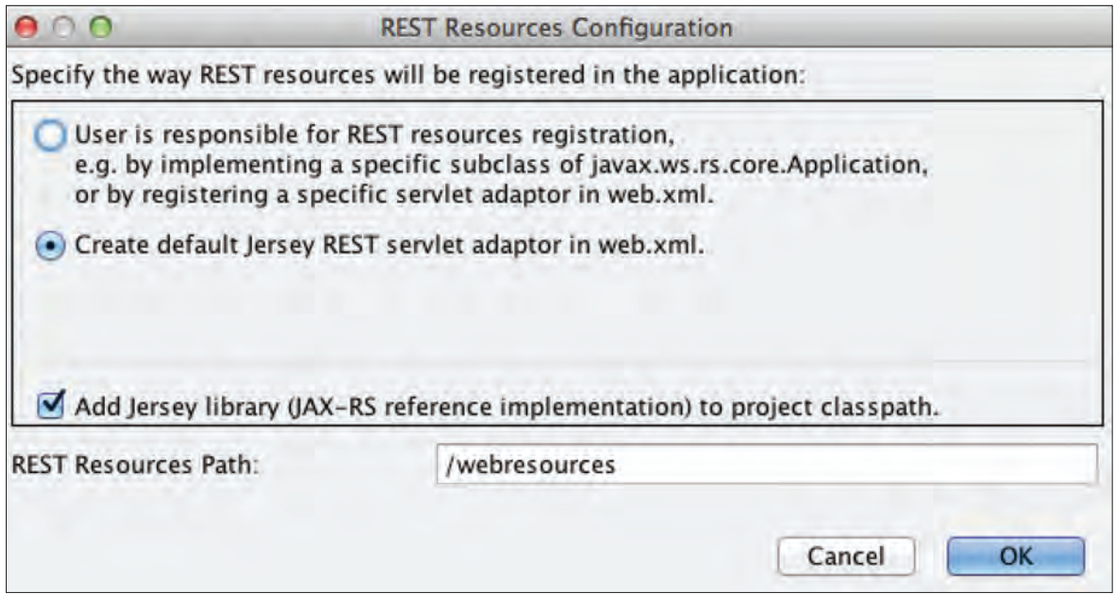


図5

らからダウンロードします。

b. ダウンロードした圧縮ファイルを、movieplex7 フォルダを配置するディレクトリに解凍します。

c. NetBeans IDE を起動し、「**File**」メニューから「**Open Project**」を選択します。

d. 解凍したディレクトリを選択し、「**Open Project**」をクリックします。このプロジェクトの構造を**図 4** に示します。

e. プロジェクトを開くと、設定ファイルを作成してアプリケーションにバンドルされた REST リソースのベース URI を設定するように求めるプロンプトが表示されます（**図 5**）。 このアプリケーションには必要な設定を記述したソース・ファイルがすでに含まれているため、「**Cancel**」をクリックしてこのダイアログ・ボックスを閉じます。

2. Maven 設定の確認：

a. **movieplex7** プロジェクトの「**Project Files**」ノードを開き、「**pom.xml**」ファイルをダブルクリックします。

b. pom.xml ファイルには、プロジェクトの依存先として javaee-api が指定されています（**図 6**）。この指定により、Java EE 7 の各種 API が Maven から取得されます。 ここで、特定のバージョン番号 7.0-b80 が指定されていることに注意してください。ダウンロードした GlassFish 4.0 ビルド（GlassFish 4.0 b80）を使用する際には、このバージョン番号を指定する必要があります。

データソースの確認とデータベース・スキーマの生成

ここでは、アプリケーションの各コンポーネントから JPA を使用してデータベースと通信する方法を詳しく見ていきます。

```
<dependencies>
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>7.0-b80</version>
  </dependency>
</dependencies>
```

図6

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="movieplex7PU" transaction-type="JTA">
    <!--
<jta-data-source>java:comp/DefaultDataSource</jta-data-source>
    -->
    <properties>
      <property
        name="javax.persistence.schema-generation.database.action"
        value="drop-and-create"/>
      <property
        name="javax.persistence.schema-generation.create-source"
        value="script"/>
      <property
        name="javax.persistence.schema-generation.create-script-source"
        value="META-INF/create.sql"/>
      <property
        name="javax.persistence.sql-load-script-source"
        value="META-INF/load.sql"/>
      <property
        name="eclipselink.deploy-on-startup"
        value="true"/>
      <property
        name="eclipselink.logging.exceptions"
        value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

図7

1. データソースの設定：

a. **movieplex7** プロジェクトの「**Other Sources**」ノードを開き、次に「**src/main/resources**」ノードを展開します。

b. 「**META-INF**」ノードを開き、「**persistence.xml**」をダブルクリックします。 デフォルトでは、ファイルは Design ビューで開きます。

c. 「**Source**」タブをクリックして XML ソース・ファイルを表示します（**図 7**）。

<jta-data-source> がコメントアウトされていることに注意してください。これはつまり、データソース要素が指定されていないということです。 jta-data-source は、基盤のアプリケーション・サーバーのランタイム環境における接続先の JDBC リソースを指定する要素です。

Java EE 7 プラットフォームでは、ランタイムにより提供されるデフォルト・データソースが新たに定義されています。 この設定済みのデータソースには、java:comp/DefaultDataSource という JNDI 名でアクセスできます。

JPA 2.1 仕様によれば、jta-data-source 要素と non-jta-data-source 要素のいずれも指定されていない場合は、デプロイ時に Java Transaction API（JTA）データソースを指定するか、デフォルトの JTA データソースがコンテナから提供される必要があります。

d. GlassFish 4 のデフォルトのデータソースは、JDBC リソースの jdbc/__default にバインドされています。 **Design** ビューと **Source** ビューをクリックして切り替える際に、**図 8** のような警告が表示されることがありますが、この警告は、アプリケーションを実行することで解消されます。「**OK**」をクリックしてこのダイアログ・ボックスを閉じます。

2. データベース・スキーマの生成：スキーマ生成用のスクリプトは META-INF ディレクトリにあります。これらのスクリプトの場所は URL で指定されているため、WAR ファイルの外部からでもこれらのスクリプトを読み込むことができます。

a. **movieplex7** プロジェクトの「**Other Sources**」ノードを開き、次に「**src/main/resources**」ノードを展開します。

b. 「**META-INF**」ノードを開き、create.sql スクリプトを実行します。**図 9** に示すデータベース・スキーマが生成されます。

JPA 2.1 では、新しい一連のプロパティ javax.persistence.schema-generation.* が導入されています。このプロパティは、表、索引、制約などのデータベース・アーティファクトをデータベース・スキーマ内に生成するために使用できます。 この機能を使用して、JPA ドメイン・オブジェクト・モデルをデータベース内に直

図8

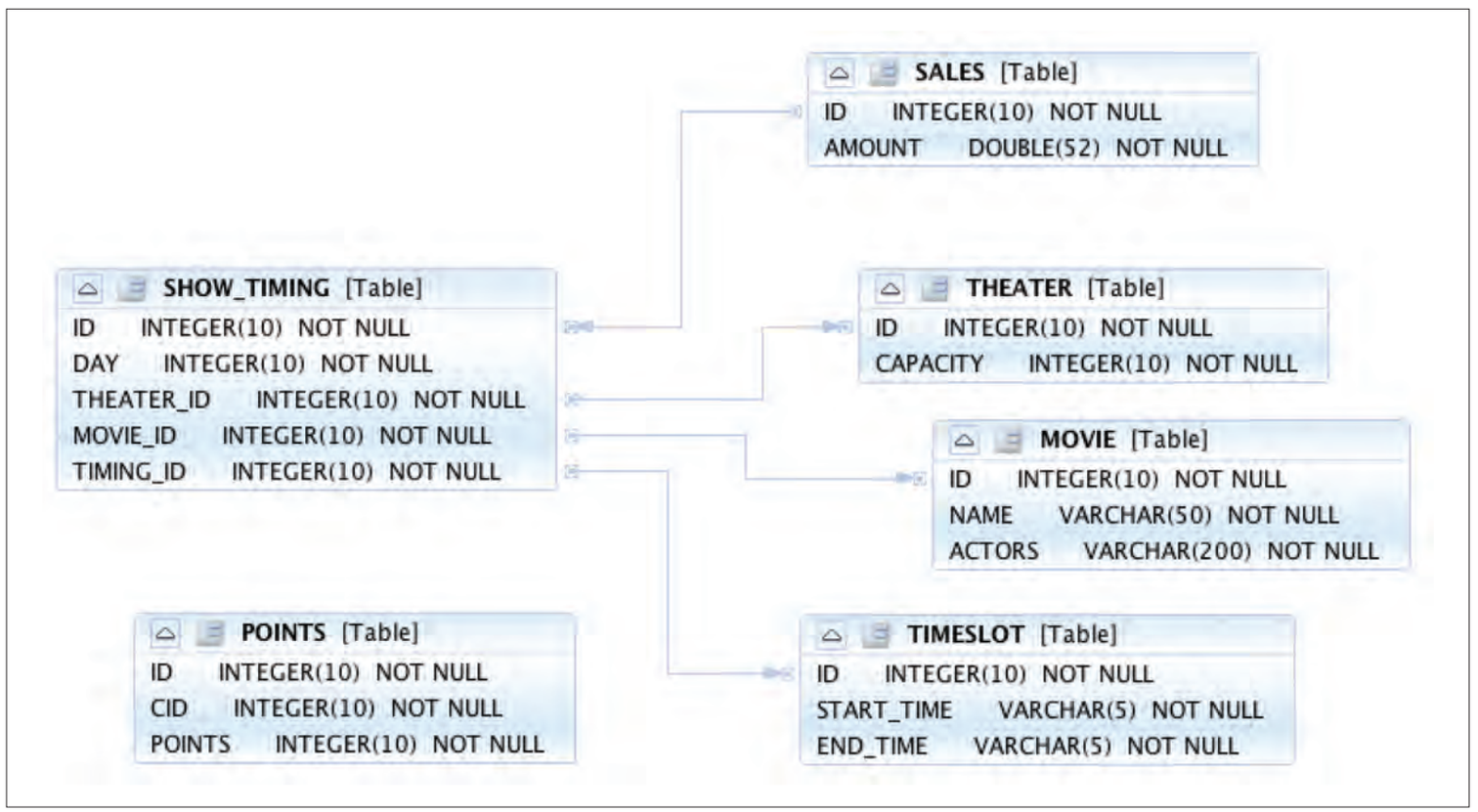


図9

接生成できます。 ただし、生成されたスキーマを本番環境で使用する場合は、微調整が必要になる場合があります。一部のプロパティについて、その意味と有効な値を**表 2** に示します。

映画を予約する Web ページの作成

このセクションでは、いずれかのスクリーンでの特定の上映を予約できる Web ページを作成します。 JSF のバージョン 2.2 で導入された新機能を利用します。

JSF 2.2 で新たに導入された Faces Flow は、アプリケーションで定義されたエントリ・ポイントと終了ポイントにより、関連するビュー / ページをカプセル化する機能です。 Faces Flow の中心的な考え方は、Oracle ADF タスク・フロー、Spring Web Flow、Apache MyFaces CODI から流用されています。

このサンプル・アプリケーションでは、ユーザーが映画を予約するためのフローを構築します。 このフローには 4 つのページが含まれます。各ページの目的は次のとおりです。

- 予約（Booking）ページ：映画の一覧表示
- 上映時刻表示（Show Times）ページ：選択した映画の予約可能な上映時刻の一覧表示
- 確認（Confirmation）ページ：選択した映画、上映時刻、スクリーンの予約情報作成
- 予約結果（Print/Reservation Confirmed）ページ：予約情報の表示

また、WEB-INF/template.xhtml ファイルを使用して、ヘッダー、左側のナビゲーション・バー、メイン・コンテンツ・セクションから成る Web ページのテンプレートを定義します。 このファイルは WEB-INF ディレクトリ内に置かれているため、テンプレートにはアプリケーションにバンドルされたページからのみアクセスできます。このファイルが WEB-INF ディレクトリ外に置かれている場合は、アプリケーションの外部からアクセスできるため、他の外部ページでもこのテンプレートを使用できるようになります。

1. 予約（Booking）ページの作成：
   a. NetBeans IDE で、**movieplex7** プロジェクトの「**Web Pages**」ノードを右クリックし、「**New**」を選択して、「**Folder**」を選択します。
   b. **Folder Name** フィールドに booking と入力し、「**Finish**」をクリックします。
   c. 新しく作成したフォルダを右クリックし、「**New**」を選択して、「**Other**」を選択します。
   d. ダイアログ・ボックスの **Categories** セクションで「**JavaServer Faces**」を選択し、**File Types** セクションで「**Facelets Template Client**」を選択します（**図 10**）。 次に、「**Next**」をクリックします。
   e. **File Name** フィールドに booking と入力します。
   f. **Template** の横の「**Browse**」をクリックし、**WEB-INF** ノードの下の「**template.xhtml**」ファイルを選択します。
   g. 「**Finish**」をクリックします。
2. booking.xhtml ファイルの編集：
   a. <ui:define> セクションを**リスト 1** のように変更します。

      リスト 1 のコードにより、映画の一覧をラジオ・ボタンで表示する HTML フォームが作成されます。 選択した映画は #{booking.movieId} にバインドされ、フローをスコープとする Bean として定義されます。 commandButton の action 属性の値は、フローにおける次のビュー（このサンプルでは、同じディレクトリ内にある showtimes.xhtml）を表します。
   b. ヒント（黄色の電球）をクリックし、表示される候補をクリックして名前空間の接頭辞を追加します。 同じ操作を、接頭辞 c: と f: に対しても実行します（**図 11**）。
3. 予約 Java Bean の作成とフローとしての定義：
   a. **movieplex7** プロジェクトの「**Source Packages**」ノードを右クリックし、「**New**」を選択して、「**Java Class**」を選択します。
   b. **Name** フィールドに Booking と入力し、さらに **Package Name** フィールドに org.glassfish.movieplex7.booking と入力します。
   c. クラスレベルのアノテーションである @Named を追加して、このクラスに式言語（EL）をインジェクションできるようにします。

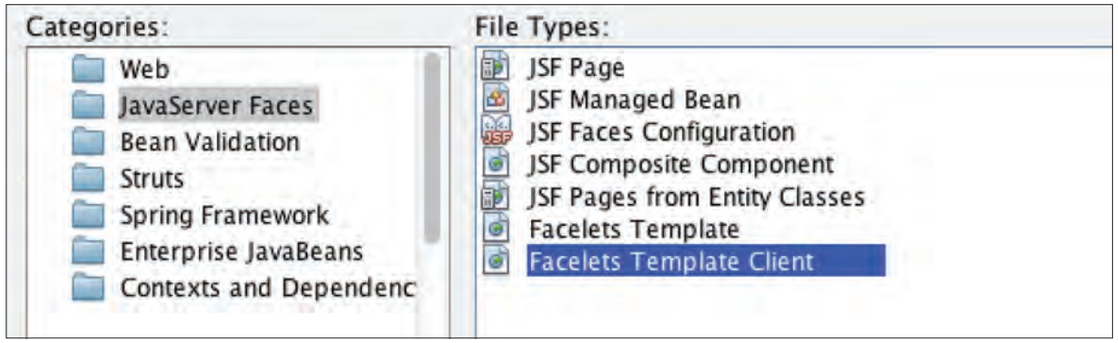


図10

プロパティ	説明	値
javax.persistence.schema-generation.database.action	データベース・アーティファクトに関して永続化プロバイダが実行するアクションを指定	none、create、drop-and-create、drop
javax.persistence.schema-generation.create-source	データベース・アーティファクトの作成において、オブジェクト・リレーショナル・マッピング・メタデータ、データ定義言語（DDL）スクリプト、あるいはこれら 2 つの組み合わせのいずれを使用するかを指定	metadata、script、metadata-then-script、script-then-metadata

表2

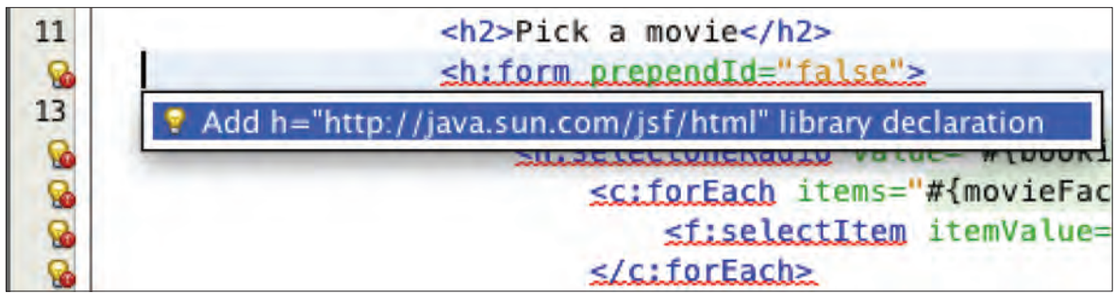


図11

d. @FlowScoped("booking") を追加して、Bean のスコープをフローとして定義します。
この Bean は自動的に、フローの開始時にアクティブ化され、フローの終了時に非アクティブ化されます。

int movieId プロパティを Booking.java ファイルに追加します。

e. 編集しているソース・コード内を右クリックして、getter/setter を生成します。
f. 「**Insert Code**」、「**Getter and Setter**」の順に選択し、フィールドを選択します。
g. public String getMovieName() メソッドを**リスト 2** のように追加します。
このメソッドは、選択された映画の題名を返します。
h. 次のコードを追加して、このクラスに EntityManager をインジェクションします。

```
@PersistenceContext
EntityManager em;
```

または、選択されたラジオ・ボタンより映画の ID と題名を渡して、バッキング Bean で解析することもできます。この方法を使用すれば、データベースとの余分な通信を抑制できます。

4. 上映時刻表示（Show Times）ページの作成：
a. 手順 1 と 2 を繰り返して、<ui:define> セクションの内容を**リスト 3** のように変更します。
リスト 3 のコードにより、選択された映画の題名とすべての上映時刻を表示する HTML フォームが作成されます。#{timeslotFacadeREST.all} はすべての映画の一覧を返します。この一覧に対して、c:forEach ループで反復処理を行います。 選択された上映の ID と開始時刻は、#{booking.startTime} にバインドされます。 (value="Back") コマンド・ボタンは前のページに戻るためのボタンであり、(value="Confirm") コマンド・ボタンはフローにおける次のビュー（このサンプルの場合は confirm.xhtml）に移動するためのボタンです。
通常、ユーザーは選択した映画の上映時刻のみが表示されることを期待しますが、このページにはすべての上映時刻が表示されます。 このサンプルでは、誤った上映時刻を選択した場合にフロー内を前後に移動する状況を示すためにこのような仕様になっています。

5. Booking.java ファイルの修正：
a. フォームから受信した値を解析するためのプロパティとメソッドを追加します（**リスト 4**）。
b. さらに、選択された映画と上映時刻で予約可能な最初のスクリーンを検索するためのメソッドも追加します（**リスト 5**）。
6. 確認（Confirmation）ページの作成：
a. 手順 1 と 2 を繰り返して、<ui:define> セクションの内容を**リスト 6** のように変更します。
リスト 6 のコードにより、選択された映画と上映時刻、およびスクリーン（予約できるものがある場合）が表示されます。 これら 3 つの情報がすべて存在する場合のみ、予約を続行できます。 print.xhtml はフローの最後のページであり、確定された予約情報を表示します。このページは、commandButton の action の値が Book であることにより識別されます。
actionListener を commandButton に追加して、予約を行うためのビジネス・ロジッ

リスト1　リスト2　／リスト3　／リスト4

```
<ui:define name="content">
  <h2>Pick a movie</h2>
  <h:form prependId="false">

    <h:selectOneRadio value="#{booking.movieId}"
layout="pageDirection" required="true">
      <c:forEach items="#{movieFacadeREST.all}" var="m">
        <f:selectItem itemValue="#{m.id}" itemLabel="#{m.name}"/>
      </c:forEach>
    </h:selectOneRadio>

    <h:commandButton id="shows" value="Pick a time"
action="showtimes" />}"/>
  </h:form>
</ui:define>
```

すべてのリストのテキストをダウンロード

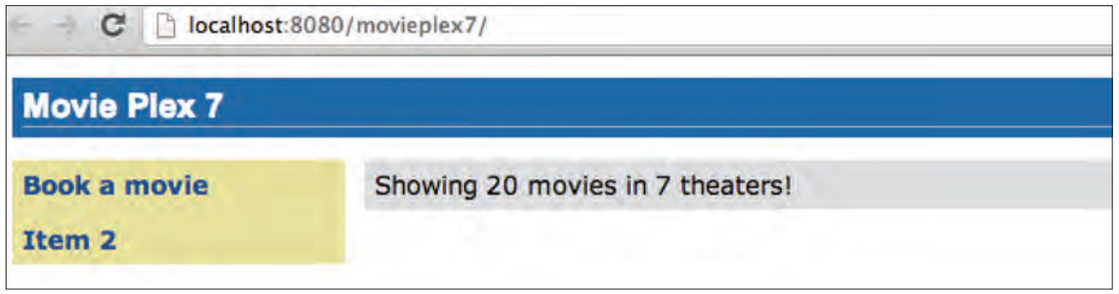


図12

クを呼び出すことができます。

7. 予約結果（Print/Reservation Confirmed）ページの作成：
a. 手順 1 と 2 を繰り返して、<ui:define> セクションの内容を**リスト 7** のように変更します。
リスト 7 のコードにより、映画の題名、上映時刻、選択されたスクリーンが表示されます。 この commandButton はフローの終了ポイントであり、このボタンを押すとアプリケーションのホームページに戻ります。
8. フローの設定：
すべてのページ（booking.xhtml、showtimes.xhtml、confirm.xhtml、print.xhtml）が同じディレクトリ内にあります。そのため、このディレクトリ内のビューをフロー内のビュー・ノードとして扱うことをランタイムに通知する必要があります。
a. **Web Pages/booking** フォルダを右クリックし、「**New**」を選択して、「**Other**」を選択します。
b. 「**XML**」を選択し、「**XML Document**」を選択します。
c. **Name** フィールドに booking-flow と入力し、「**Finish**」をクリックします。
d. 生成された XML ファイルの booking-flow.xml を、**リスト 8** のように変更します。
9. template.xhtml ファイル内のフローの呼び出し：WEB-INF/template.xhtml ファイルを編集し、**リスト 9** に示すコードの commandLink を**リスト 10** のように変更します。

commandLink は、フォームの送信ボタンのように動作する HTML アンカー・タグにレンダリングされます。 action 属性には、フローのすべてのビューを格納しているディレクトリを指定します。 このディレクトリにはすでに booking-flow.xml があり、このファイル内でページのフローを定義しています。

サンプル・アプリケーションのテスト

次に、この映画予約アプリケーションをテストします。 ここでは、作成したすべてのページを呼び出します。

1. 「**movieplex7**」ノードを右クリックし、「**Run**」を選択してプロジェクトを実行します。
ブラウザに更新後の画面が表示されます（**図 12**）。

リスト5 リスト6 リスト7 リスト8 リスト9 リスト10

```
public String getTheater() {
 // for a movie and show
 try {
  // Always return the first theater
  List<ShowTiming> list =
      em.createNamedQuery
("ShowTiming.findByMovieAndTimingId",
       ShowTiming.class)
     .setParameter("movieId", movieId)
     .setParameter("timingId", startTimeId)
     .getResultList();
  if (list.isEmpty())
   return "none";
  return list
   .get(0)
   .getTheaterId()
   .getId().toString();
 } catch (NoResultException e) {
  return "none";
 }
}
```

すべてのリストのテキストをダウンロード

2. 「**Book a movie**」をクリックして予約ページを表示します（**図 13**）。
3. 映画を選択し（例：The Shining）、「**Pick a time**」をクリックして、上映時刻表示ページを表示します（**図 14**）。
4. 時刻を選択し（例：04:00）、「**Confirm**」をクリックして、確認ページを表示します（**図 15**）。
5. 「**Book**」をクリックして予約を確定し、予約結果ページを表示します（**図 16**）。

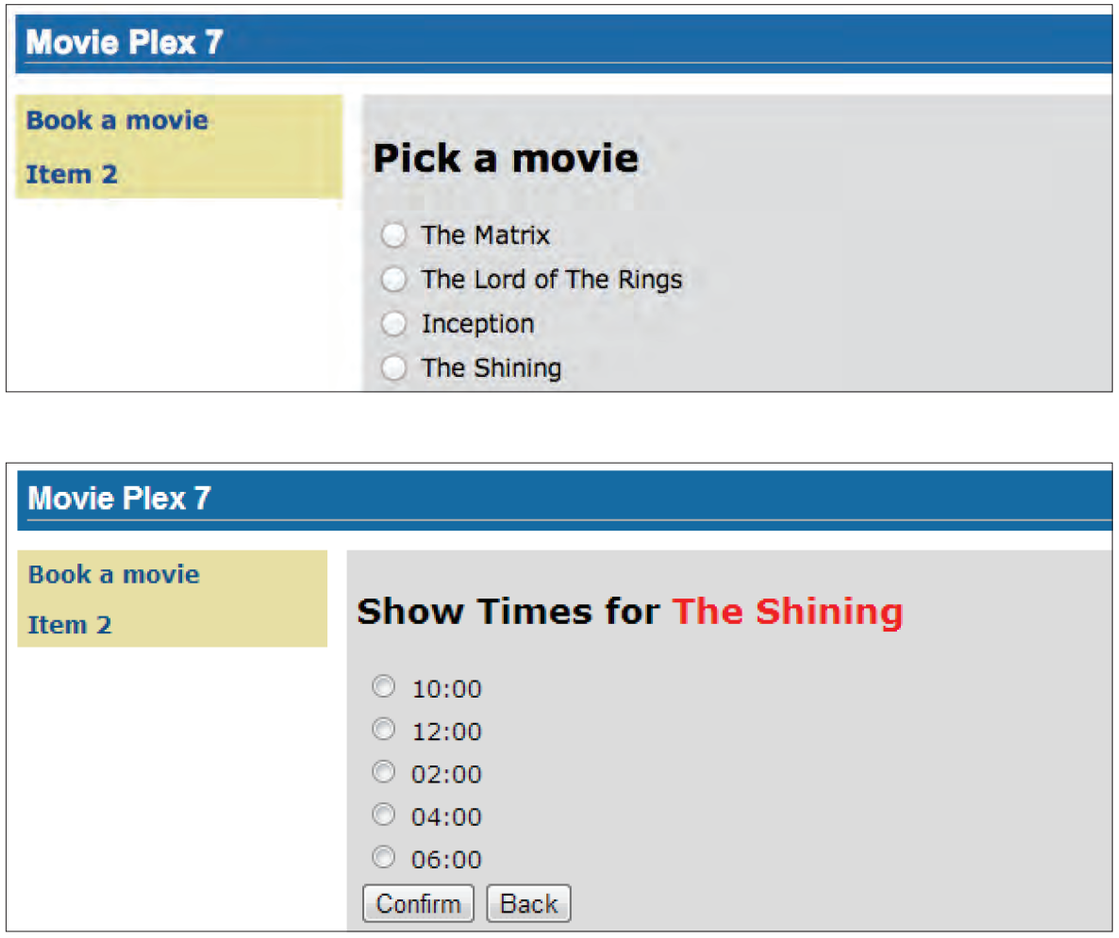
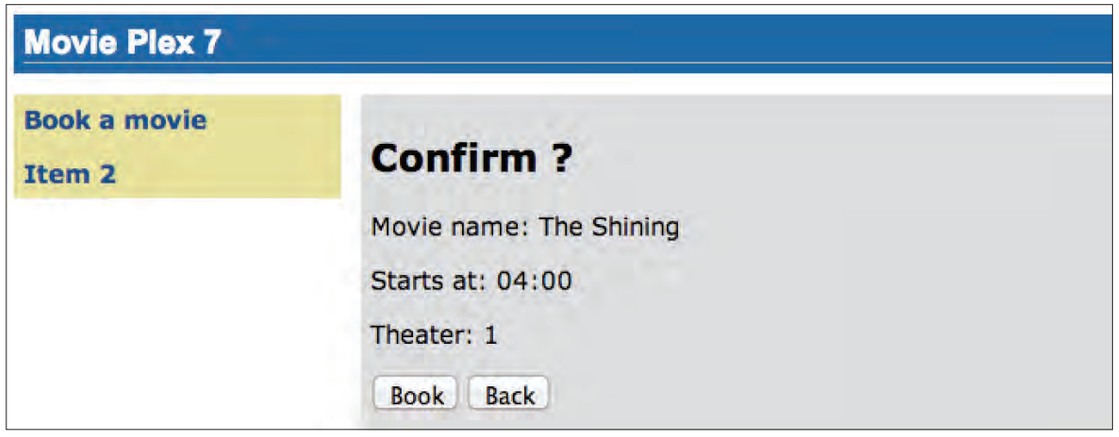


図14



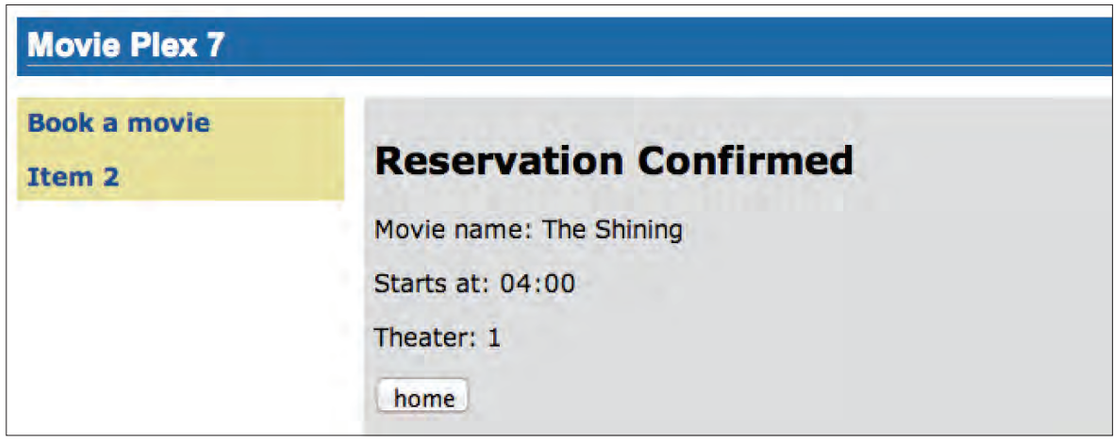


図16

まとめ

Java EE 7 には多数の興味深い新機能があります。 本記事では、Java EE 7 に搭載されている次のテクノロジーを使用して、3 層 Web アプリケーションを作成しました。

- Java Persistence API 2.1（JSR 338）
- JavaServer Faces 2.2（JSR 344）
- Contexts and Dependency Injection 1.1（JSR 346）

サンプルの movieplex7 Web アプリケーションは、これらの新規 API がそれぞれ特定のニーズに対応するように設計されています。たとえば、本記事では JavaServer Faces をユーザー・インタフェースの実装に使用し、Enterprise JavaBeans と Java Persistence API を「予約表示」機能の実装に利用しました。

このシリーズの次回以降の記事では、movieplex7 Web アプリケーションを改良して、4 つの新機能（映画の追加 / 削除、チケット販売、映画ポイントの付与、チャット・ルーム）を追加します。その際、Batch Applications for the Java Platform 1.0（JSR 352）、Java API for JSON Processing 1.0（JSR 353）、Java API for WebSocket 1.0（JSR 356）、Java Transaction API 1.2（JSR 907）、JAX-RS 2.0（JSR 339）などの、Java EE 7 に搭載されている重要なコンポーネントや API を利用します。 **</article>**

詳細

- GlassFishのリソース
- 「Java EE 7—Introduction」
- 「Java EE 7 Hands-on Lab」
- Java EE Platform仕様

JULIEN PONGE

BIO

Julien Ponge（@jponge）：長年の経験を有するオープン・ソース・プロフェッショナルで、現在はフランス国立応用科学院リヨン校のコンピュータ・サイエンスおよびエンジニアリング准教授を務める。 CITI Laboratory では、プログラミング言語、仮想マシン、ミドルウェアについて研究している。

写真：
MATT BOSTOCK/GETTY IMAGES

パート2
invokedynamic の神秘を解く

invokedynamicの詳細を習得する方法

パート 1 では、invokedynamic の基本を取り上げ、メソッド・ハンドルを取得する方法や、実行時にメソッド・ハンドルをコール・サイトにバインドする方法、実際の invokedynamic の命令を含む Java 仮想マシン（JVM）バイトコードを生成する方法について説明しました。 ただし、invokedynamic の概要を紹介したのみで、java.lang.invoke API の詳細については触れませんでした。

特に、パート 1 で作成したメソッド・ハンドルのコンビネータ・チェーンは、ほぼ直線的なものでした。 パート 1 では、ターゲット・メソッドへのメソッド・ハンドルを使用し、型の異なるコール・サイトとターゲットを適応させるために、引数の個数や型の変換を何度か実行しました。たとえば、(Object, Object)Object 型のコール・サイトを (int, int[])int 型のメソッド・ハンドルに適応させました。

本記事では、より複雑なロジックをメソッド・ハンドル・チェーンに組み込めることを確認していきます。 具体的には、値のフィルタリング、条件分岐、例外のキャッチができることを確認します。 重要な点は、メソッド・ハンドルを組み合わせることでプログラミング・ロジックの大半を作成できるということです。 実際、正しく使用すれば MethodHandle API は万能で、明示的にバイトコードを使用しなければならない状況から開発者を解放してくれます。 本当に強力な API なのです。 本記事では、該当するコンビネータについてそれぞれ単純な例を示し、本物の動的言語を実装するという観点から各コンビネータの使用方法を説明します。

型の操作

JVM 上で稼働する言語を実装するとなると、特に Java との相互運用性を確保する目的で、ただちに型の変換が必要になります。 幸いにも、変換は多くの場合、フィルタリング用のコンビネータを使用するだけで解決できます。

ここで、**リスト 1** に示す RichInt クラスについて考えます。

RichInt クラスは、プリミティブの int 型をラップし、さらに square（2 乗）メソッドが追加されています。 言うまでもなく、言語の世界における「強化型整数」の完全な実装には square メソッド以外の機能も含まれますが、リスト 1 の実装は基本的な考え方を示すためのものです。

リスト 2 のように、square のメソッド・ハンドルを取得し、RichInt のインスタンスをレシーバとして渡すことで、このメソッド・ハンドルを呼び出すことができます。 しかし、代わりに int の値を渡し、それらの int の値に対して square を呼び出し、戻り値として int 値を取得する方が便利ではないでしょうか。

この種の変換は、MethodHandles クラスのコンビネータである filterArguments と filterReturnValue を使用することで実現できます。 filterArguments は、引数リスト内の位置、およびメソッド・ハンドルの可変長リストを引数に取ります。 各メソッド・ハンドルは、指定された位置の引数を先頭に、各引数をフィルタリングします。 filterReturnValue の動作もほぼ同じですが、引数のメソッド・ハンドルは 1 つのみです。

このメソッド・ハンドルの呼出しには、invokeExact を使用します。 invokeExact では、引数と戻り値の型がメソッド・ハンドルのものと完全に一致している必要があります。

しかし、型がコンパイル時には分からず、引数と戻り値の型を一致させることが不可能な場合もあります。そのような場合は invoke や invokeWithArguments を代わりに使用できます。

invokeExact メソッドとの違いは、invoke メソッドや invokeWithArguments メソッドでは型の変換（プリミティブ型のボクシングとアンボクシングなど）を実行する必要があることです。 可能な限り、パフォーマンスに優れている invokeExact を使用することをお勧めします。

RichInt の新しいインスタンスの作成時に、int から RichInt への変換が実行されます。 また、RichInt の仮想メソッドである intValue を呼び出した場合は、この逆に RichInt から int への変換が実行されます。 **リスト 3** のコードに、この変換の方法を示します。

変換を行うことで、int を RichInt のように扱い、square メソッドを呼び出して int を返すことができます。

guardWithTest を使用した分岐

動的言語を実装する際はしばしば、コール・サイトで if/then/else の条件分岐をマップする必要が生じます。条件分岐が必要となるのは、たとえば、レシーバの型や何らかの引数の値に基づいてメソッド呼出しを異なるターゲットにディスパッチする場合などです。 パート 1 では直線的なコンビネータ・チェーンのみを紹介しましたが、直線的ではないコンビネータ・チェーンも実装可能です。

MethodHandles クラスには guardWithTest というメソッドがあります。このメソッドは、条件分岐を含むコンビネータを作成するために使用します。 そのためには、guardWithTest に以下の 3 つのメソッド・ハンドルを入力値として渡します。

- ガード条件を実装して Boolean 型の値を返す、フィルタ条件用のメソッド・ハンドル
- ガードにより true が返されたときに実行されるメソッド・ハンドル
- ガードにより false が返されたときに実行されるメソッド・ハンドル

ガード用のメソッド・ハンドルはコンビネータ・チェーンで渡される引数を使用しますが、渡されるすべての引数を使用するとは限りません。 たとえば、最初のパラメータのみに基づいて条件を評価することもできます。 一方、true/false 式の分岐を行うメソッド・ハンドルでは、すべての引数を処理する必要があります。この際には、直接処理するか、さらにコンビネータを使用して適応を行った上で処理することもできます。

それでは、例となるコードを Guards というクラスで作成しましょう。 この例では、(Object)String シグネチャのコール・サイト向けに、メソッド・ハンドルのコンビネータ・チェーンを作成します。このコール・サイトでは、オブジェクトを渡して、そのオブジェクトの文字列表現を取得します。 Integer 型および String 型の引数については特化された表現を取得し、その他の型については汎用の表現を取得します。 そのために、3 つの静的メソッドを作成します（**リスト 4**）。

次に、guardWithTest を利用し、引数のクラスに応じてリスト 4 の 3 つのメソッドのいずれかをターゲットとする分岐構造を作成します。

3 つのケースがあるため、**リスト 5** のような 2 つのガード・メソッドが必要になります。

リスト 5 では、value が null ではないことを前提としています。 instanceof 演算子を使用することもできますが、型の階層が深くなりパフォーマンスが低下する可能性があります。そのため、参照を getClass() と比較すること（さらに、null の可能性がある場合は null とも比較すること）が、instanceof によるチェックを行わずに済むもっとも確実な方法です。 テスト・メソッドを**リスト 6** に示します。

リスト 6 では、最初にルックアップ・オブジェクトを取得して、次にターゲット・メソッドとガード・メソッドに対するメソッド・ハンドルを取得しています。

リスト1 リスト2 ／リスト3 ／リスト4 ／リスト5 ／リスト6

```
public class RichInt {

  private final int value;

  public RichInt(int value) {
    this.value = value;
  }

  public int intValue() {
    return value;
  }

  public RichInt square() {
    return new RichInt(value * value);
  }

  @Override
  public String toString() {
    return "RichInt{" + "value=" + value + "}";
  }
}
```

すべてのリストのテキストをダウンロード

ターゲット・メソッドのシグネチャはそれぞれ異なるため、asType コンビネータを使用して各メソッドをコール・サイトの型、すなわち (Object)String に適応させる必要があります。 言うまでもなく、Integer や String は Object として扱えるため、問題なく適応されます。

メソッド・ハンドル・チェーンの型チェックはビルド時に実行されることにご注意ください。 したがって、asType を使用したシグネチャの適応に失敗した場合、実行時に例外が発生します。また、これも重要なことですが、プリミティブ型のボクシングとアンボクシングは、今のところ Java HotSpot VM で除去できないことがある負荷の高い処理である点も忘れないでください。

リスト 6 では最後に、条件分岐を行う branching メソッド・ハンドルを、1 つ目の guardWithTest を利用して作成しています。 オブジェクトが Integer 型の場合は intDisplay に進み、それ以外の型の場合は、2 つ目の guardWithTest の呼出しに進みます。この呼出しでは、オブジェクトが String 型の場合は stringDisplay に進み、それ以外の型の場合は汎用の objectDisplay に進みます。

このコード例を実行した場合、以下のようにコンソールに出力されます。

```
Integer = 1
String = foo
Object = true
```

実際の動的言語では、guardWithTest は、自己修正型の可変コール・サイトまたは頻繁に変更されるコール・サイトのインライン・キャッシュを構築するためによく使用されます。この考え方の起源は Smalltalk の時代にまで遡ります。簡単に説明すると、この考え方は、メソッド呼出しを行う際に、コール・サイトで検出したレシーバ・オブジェクトが同じ型である限り、ターゲットのメソッド・ハンドルをキャッシュするというものです。この方法では、呼出しのたびにメソッド・ハンドルのルックアップを実行するよりも、パフォーマンスが大幅に向上します。

インライン・キャッシュは、JVM などの適応型ランタイムでも使用できます。Just-In-Time（JIT）コンパイラでは、インタプリタ・モードとは別に、実行中のコードに関する不確実な推測に基づくネイティブ・コードが生成されます。たとえば、レシーバの型が特定の型である、あるいはマルチスレッドのコンテキストにおいてロックがまったく使用されないといった推測があります。 そのような推測は無効となることもあります。そのため、生成されたネイティブ・コードの各ブロック（「高速」パス）には、以前に最適化されたコードやインタプリタ・モード（「低速」パス）に戻すためのガードが指定されています。

インライン・キャッシュの例を本記事で取り上げるには紙幅が足りません。また、インライン・キャッシュの実装の詳細は、大部分がターゲット言語のランタイム仕様によって異なるため、これ以上の説明は割愛します。 JSR 292 Cookbook プロジェクトにおいて Rémi Forax 氏が公開している例を確認し、各言語のランタイムに適応させるテストを行うことをお勧めします。

リスト7

```
static class Runner extends Thread {

  final MethodHandle runHandle;
  final MethodHandle waitHandle;

  Runner(MethodHandle runHandle, MethodHandle waitHandle) {
    super();
    this.runHandle = runHandle;
    this.waitHandle = waitHandle;
  }

  @Override
  public void run() {
    while (true) {
      try {
        System.out.println((String) runHandle.invokeExact());
        waitHandle.invokeExact();
      } catch (Throwable e) {
        throw new RuntimeException(e);
      }
    }
  }
};
```

すべてのリストのテキストをダウンロード

スイッチ・ポイント

SwitchPoint クラスも、分岐を実装する場合に利用できる便利な機能の 1 つです。特に、「特定の条件」を満たす場合にターゲットを実行し、その条件が満たされなくなった場合は永久的にフォールバック・ターゲットに戻すような場合に使用できます。

典型的なケースは、オブジェクトのインスタンス内のメソッドを実行時に置き換えることのできる動的言語です。 スイッチ・ポイントはメソッドが変更されない限りは有効状態を維持し、メソッドが置き換えられた後は無効化されます。guardWithTest を使用した構造とほぼ同じですが、変更されることのない同じ条件に多数のコール・サイトが依存している場合は、スイッチ・ポイントを使用する方が簡単です。

リスト 7 に、2 つのメソッド・ハンドルを引数として受け取る実行スレッドの例を示します。 このコードの無限ループには、()String 型のメソッド・ハンドルである runHandle が含まれます。

runHandle は、呼び出されるたびに文字列値を出力します。 waitHandle は () void 型であり、ある一定時間スレッドをスリープ状態にします。

テスト・メソッドではこのスレッドを以下の 2 つのフェーズで実行します。

- runHandle で約 5 秒間、500 ミリ秒ごとに「a」を返す
- その後、runHandle で 1,200 ミリ秒ごとに「b」を返す

この処理を行うために、まずは**リスト 8** の静的メソッドを定義します。これらの静的メソッドは、後でメソッド・ハンドルのターゲットとして使用します。 テスト・メソッドを**リスト 9** に示します。

リスト 9 では、まずは通常どおり、ターゲット・メソッドのメソッド・ハンドルを作成しています。 次に、2 つのスイッチ・ポイントをインスタンス化しています。 それぞれ、実行ターゲットに対応するインスタンスと、待機ターゲットに対応するインスタンスです。

SwitchPoint の guardWithTest メソッドを呼び出すことで、メソッド・ハンドルを取得できます。 前述の guardWithTest コンビネータを直接使用する方法とよく似ていますが、今回は 2 つのメソッド・ハンドルのみを引数として受け取っています。1 つ目は、スイッチ・ポイントが有効な間に実行されるメソッド・ハンドルで、2 つ目は、スイッチ・ポイントが無効化された直後に実行されるメソッド・ハンドルです。

SwitchPoint が呼び出されるたびに実行されるような明示的なガード・コードはありません。 同じセマンティクスを、何らかの参照オブジェクトや式に対して評価されるガードを含む guardWithTest を使用して実現することもできます。

この後、スレッドにスイッチ・ポイントのターゲット用の各メソッド・ハンドルを渡して、スレッドを開始できます。 次に、このスレッドを 5 秒間スリープ状態にして、さらにその後、SwitchPoint クラスの静的メソッドである invalidateAll を使用して、これら 2 つのスイッチ・ポイントを無効化しています。 この時点で、アプリケーションが終了するまで、1,200 ミリ秒ごとに「b」が出力されるようになります。

一連のスイッチ・ポイントを無効化することで、それぞれのターゲットの分岐がすべて切り替えられます。 分岐の切替えは、頻繁に変更されるセマンティクスを持つ同時実行中のスレッドすべてに即座に反映されます。 負荷の高い処理となる可能性はありますが、無効化された直後に変更内容を全体に適用することのみを目的とする場合は、SwitchPoint を MutableCallSite と組み合わせて使用する方が、VolatileCallSite のみを使用するよりも効率的です。 コール・サイトが無効化される割合に対してパフォーマンス分析を実行することで、最適な選択を行うことができます。

SwitchPoint と guardWithTest 呼出しのどちらを選択すべきかは、無効化の実行方法によっても異なります。 無効化を実行するかどうかをレシーバの型または引数のみから判断する場合は、guardWithTest を使用すべきです。 無効化の判断はローカルでの呼出し時に行われ、たとえば、インライン・キャッシュの呼出しに適用されます。

リスト8 リスト9

```
static String a() {
  return "a";
}

static String b() {
  return "b";
}

static void slow() {
  try {
    Thread.sleep(1200);
  } catch (InterruptedException e) {
    throw new RuntimeException(e);
  }
}

static void fast() {
  try {
    Thread.sleep(500);
  } catch (InterruptedException e) {
    throw new RuntimeException(e);
  }
}
```

すべてのリストのテキストをダウンロード

一方、クラス定義のメソッドだけではなくオブジェクトのメソッドも変更できるような動的言語のケースについてはどうでしょう。 Ruby プログラミング言語がこのような動的言語の良い例です。Ruby ではクラスを以下のように定義できます。

```
class Foo
  def bar
    "bar"
  end
end

foo = Foo.new
puts foo.bar # prints bar
```

この例で呼び出されるメソッドは、Foo クラスのインスタンスである foo オブジェクトの bar メソッドです。 さらに、Ruby では以下のように foo インスタンス上の bar を再定義することもできます。

```
def foo.bar
  "[bar]"
end
puts foo.bar # prints [bar]
```

このような場合、bar を再定義した例

の bar の呼出し部分などのように、特定のメソッドが有効なままであることに依存しているコール・サイトが多数あります。 無効化は、インスタンス・メソッドが変更されたときに実行されます。 インスタンス・メソッドの変更は、guardWithTest を使用し、さらにインスタンス・レベルの条件をガードとして使用することで実行できますが、SwitchPoint の方が優れています。 頻繁に変更されるセマンティクスにおいては、すべてのコール・サイトを一度に無効化でき、さらに興味深いことに、無効化が不要である限り事実上 SwitchPoint のオーバーヘッドが発生しないためです。

それほど例外的ではない例外

メソッド・ハンドルのターゲットは、通常の Java メソッドと同様に例外をスローできます。 コール・サイトは実行時に動的にバインドされるため、ターゲットで例外が発生しているにもかかわらず、invokedynamic の命令周辺のバイトコードでその例外がスローされることを想定していないという状況に陥ることがあります。

MethodHandles クラスは、catchException と呼ばれる、try/catch 構造をメソッド・ハンドルのレベルで再現可能なコンビネータを提供しています。 catchException コンビネータは、通常のターゲットへのメソッド・ハンドル、例外クラス、catch ブロックと同様の働きを持つ呼出し対象のメソッド・ハンドルを引数として受け取ります。 同様に、MethodHandles クラスには、コール・サイトで例外をスローするために使用できる throwException コンビネータもあります。

本記事の例では、Groovy プログラミング言語のセーフ・ナビゲーション演算子（?.）を参考にしました。この演算子を使用して、メソッド・コールをチェーンとして連結したときに悪名高い NullPointerException が発生するのを防ぐことができます。 実は、チェーンの途中のメソッドが null を返す場合、次の呼出しで NullPointerException が発生します。 そこで、以下のようにセーフ・ナビゲーション演算子をメソッド呼出しの演算子として配置することで、この問題を防ぎ、代わりに null を返すようにします。

```
def mrbeanPhone =
person?.friends["mrbean"]?.phone
```

friends が null の場合、または「mrbean」エントリが存在しない場合に、mrbeanPhone は null になります。 このコードの演算子を単純な「.」に置き換えると、そのような状況では NullPointerException が発生します。

それでは、通常のターゲットで例外がスローされた場合に値を返す構造に対応するようなランタイム・サポートを作成します。**リスト 10** のターゲット・メソッドを見てください。

このコードでは明らかに、obj パラメータに null を渡した場合に NullPointerException が発生します。 そこで、例外が発生した場合に、**リスト 11** に示すターゲットのメソッド・ハンドルを使用して、別の表現を返すことにします。リスト 11 は、引数がすべて 1 つの配列にラップされている場合に、あらゆるコール・サイトのシグネチャで動作します。 また、catchException も利用できます（**リスト 12**）。

リスト10　リスト11　／リスト12

```
static String stringAndHashCode(String prefix, Object obj) {
  return new StringBuilder(prefix)
         .append(obj.toString())
         .append(" #")
         .append(obj.hashCode())
         .toString();
}
```

すべてのリストのテキストをダウンロード

stringAndHashCode へのメソッド・ハンドルは簡単です。

一方、npeHandler へのメソッド・ハンドルでは、中間的なコンビネータを使用して適応を 2 回行う必要があります。最初の適応ですべての引数を 2 つの引数による配列にまとめ（asCollector）、2 回目ではコール・サイトの型の適応を行います。 その後は、catchException により、NullPointerException が発生しない場合にメソッド呼出しを適切に stringAndHashCode にディスパッチするメソッド・ハンドルが生成されます。

プログラムを実行すると以下のように出力されます。

```
>>> 666 #666
>>> Hey! #2245797
[null] -> [>>> , null]
[null] -> [null, null]
```

興味深い点は、コンビネータを使用し

て例外処理と例外のスローを実行できるところです。 関連するバイトコード・レベルのサポートは求められません。ちなみに、バイトコード・レベルでサポートする場合は、try によるロックの範囲を区切り、例外の型および catch ブロックへの制御のジャンプを指定する必要があるため、複雑になります。 また、複数の catch 節を使用する場合は、このような呼出しをバイトコード内でネストする必要があります。 一方、コンビネータを使用した解決策では、バイトコードや関数の合成が単純になります。

パフォーマンスに関する疑問も思い浮かぶでしょう。 例外がスローされるのを待つのではなく、guardWithTest を使用して null 値をチェックすれば良いのではないか、という疑問です。 あらゆる状況で言えることですが、例外の予想発生頻度によって、それぞれの解決策の価値も変わります。 例外は、その名のとおり頻度が低いものであるべきです。 例外の頻度が低い場合は、通常のケースではターゲットのメソッド・ハンドルに高速にディスパッチし、例外がスローされる発生頻度の低いケースでは低速にディスパッチするという方法を試すことができます。 利用する解決策については必ずベンチマーク評価を行ってください。

ロジックの処理メソッド・ハンドル・チェーンには複雑なロジックを組み込むことができます。たとえば、値のフィルタリング、条件分岐、例外のキャッチが可能です。

まとめ

本記事で、invokedynamic のシリーズは終了です。 java.lang.invoke API の一部を取り上げ、メソッド・ハンドルを使用して、より複雑なロジックを作成しました。 これで、皆さんがご自身で invokedynamic をさらに学習し、調査していく準備が整いました。 invokedynamic はかなり低レベルの API ですが、適応型の最適化を実施する場合に、JVM に悪影響を及ぼさずに遅延バインドをサポートするために必要となるあらゆる機能を備えています。

最後に重要なこととして、共通の実行時基盤ロジックを提供する Dynalink リンカーなどの高レベルのライブラリも非常に便利です。 実のところ、java.lang.invoke API を使用した動的言語用のランタイム・サポートの開発は決して簡単ではありません。 Dynalink では、既製の効率的なガードを伴う呼出し、型の変換、オーバーロードされたメソッドの解決、可変引数処理など、すぐに使用できるさまざまな機能ブロックが提供されています。 また、Dynalink はメタオブジェクト・プロトコル・ライブラリも提供しています。このライブラリは、Dynalink が支える JVM 言語（オラクルの Nashorn など）と Java の相互運用性を大幅に改善します。

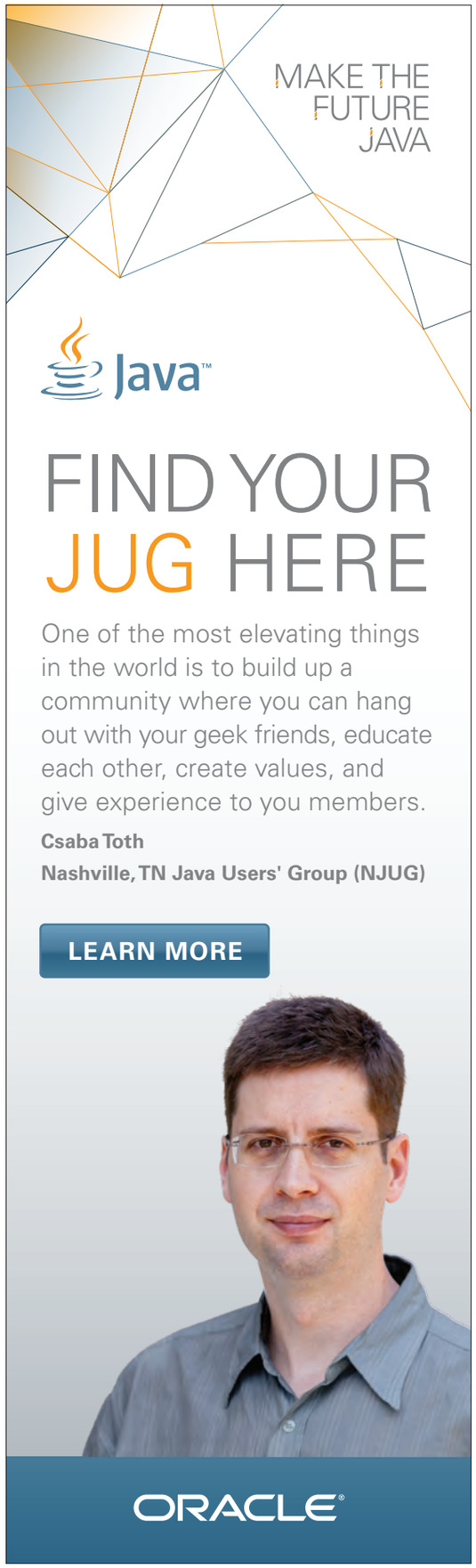
JVM 対応の既存の動的言語について、invokedynamic をサポートするためのコードを提供していただける方や、これから独自の言語を設計する方、また、invokedynamic を使用したプログラミングを楽しみたい方もいらっしゃるでしょう。 どうか歩み続けてください。その先の未来は想像できるはずです。

謝辞：レビュー段階で思慮に富む建設的なフィードバックをくださった Marcus Lagergren 氏に心から感謝いたします。

</article>

LEARN MORE

- John R. Rose：『Bytecodes Meet Combinators: invokedynamic on the JVM』
- Julien Ponge、Frédéric Le Mouël：『JooFlux: Hijacking Java 7 InvokeDynamic to Support Live Code Modifications』
- Rémi Forax：『JSR 292 Cookbook』（一般的な invokedynamic パターンの一覧）
- Attila Szegedi：『Dynalink: Dynamic Linker Framework on the JVM』



BEN EVANS、
MARTIJN VERBURG

Ben Evans（@kittylyst）：jClarity の CEO であり London Java Community（LJC）主催者。Java SE/EE Executive Committee のメンバーでもある。

Martijn Verburg（@karianna）：jClarity の CTO であり LJC の共同主催者。講演経験が豊富。Verburg 氏と Evans 氏の共著に The Well-Grounded Java Developer（Manning、2012）がある。

パート3

Java 8のラムダ式を使用した高度な操作

遅延評価と並列操作によって、かつては不可能であった最適化への道が開ける

本記事は、ラムダ式に関するシリーズのパート 1 とパート 2 の内容に基づいています。 Java 8 のラムダ式の基本構文やストリームのアプローチに詳しくない場合は、まず前回までの記事を読み、これらの考え方について理解することをお勧めします。

注：重要な点として、ラムダ式は現在も検討中の機能であることを念頭に置く必要があります。 そのため、古いビルドのラムダ式を使用したコード例が、新しいベータビルドでは動作しなくなる場合があります。 特に、本記事のコードでは OpenJDK Project Lambda の最新のコードを使用します。 この最新のコードにより、ラムダ式の API はまたひとつ進化を遂げています。このラムダ式の API は Oracle Early Access リリースに先立って配布されていますが、最終的に Java 8 に付属する API にかなり近いと思われます。

コレクションの限界

Java のコレクションは Java 言語に多大な貢献を果たしてきました。しかし、Java のコレクションは、すべての要素が存在し、かつメモリ内で表現されているという考えの下に成り立っています。 そのため、無限集合のような、より一般化されたデータを表現することはできません。

たとえば、すべての素数の集合は、Set<Integer> としてモデル化することはできません。すべての素数が判明しているわけではなく、また、すべての素数を表すのに十分なヒープ領域を確保できないことがその理由です。 Java 7 以前のバージョンでこのような集合をモデル化する唯一の手段は、集合の代わりに、集合を表す Iterator を操作することです。

主にイテレータと連携し、根本にあるコレクションを脇役へと追いやるような、データのビューを作成することは可能です。しかし、この方法には厳しい統制が必要であり、また、一見して Java のコレクションを操作するアプローチだとは分かりません。 Java 7 以前のバージョンでは、この種のアプローチを使用する場合、通常はコレクションの操作機能を使いやすくする外部ライブラリに頼ることになります。

今後の Java 8 のリリースでは、抽象概念の Stream インタフェースを導入することで、上記のような一般化されたデータを表すユースケースに対処します。Stream インタフェースは、基本的な有限のコレクションよりも、より一般化されたデータ構造を扱うのに適しています。 つまり、Stream は、Iterator や Collection よりも一般化された概念と考えることができます。

ただし、Stream はデータ構造ではなく、データを扱うための抽象概念です。

この違いは何を意味しているでしょうか。 その答えは、Stream は要素の保管については管理せず、また、Stream から個々の要素に直接アクセスする手段もないということです。

遅延評価と先行評価

それでは、無限数列をモデル化した場合はどうなるかについて詳しく考察します。 たとえば、次のような結果となります。

- Collection のストリーム全体を具現化できないため、collect() などのメソッドは使用できない
- 要素を Stream から引き出して操作する必要がある

- 次の要素が必要となった時点で返すコードが必要である

さらに、このアプローチを使用した場合、式の値は、その値が必要になった時点で初めて計算されます。 単にこのように書くとそれほど大したことではないように思えますが、Java にとっては大きな一歩なのです。

Java 8 より前のバージョンで式の値を計算するタイミングはすべて、式の値を変数にバインドするか関数に渡した直後でした。 この方法は先行評価と呼ばれます。先行評価は言うまでもなく、ほとんどの主要プログラミング言語における式評価のデフォルトの動作

です。

Java 8 では、Stream での遅延評価の利用という、Java にとって新たなプログラミング・パラダイムが導入されようとしています。 遅延評価は極めて強力な新機能ですが、慣れるまでに少々時間がかかります。

前回の記事で説明した、マップ / フィルタ・パイプラインについて確認しましょう（**リスト 1**）。

stream() メソッドは Stream オブジェクトを返します。 map() メソッドと filter() メソッドは、Stream のほぼすべての操作と同様に、遅延評価型のメソッドです。 また、パイプラインの最後にある collect() 操作によって、Stream 全体を Collection に具現化します。 collect() は先行評価型のメソッドです。そのため、このパイプライン全体は**リスト 2** のようになります。

つまり、Collection への具現化は別として、Stream をどこまで評価するかについてプラットフォームで完全に制御できます。そのため、Java 7 以前のバージョンでは実現できなかった幅広い最適化への道が開かれます。

遅延評価では、プログラマにはいっそうの注意が求められますが、その負担の大半はライブラリの作成者が受け持つことになります。 プラットフォームでの作業が複雑になったとしても、一般のプログラマの仕事を単純にすることが、Java 8 のラムダ式の目標です。

並列化のサポート

ストリームの遅延評価アプローチによって、ラムダ式フレームワークで並列操作をサポートできるようになります。

プリミティブ型の処理

今まで触れなかった Stream API の重要な側面の 1 つに、プリミティブ型の処理方法があります。 Java のジェネリクスでは、プリミティブ型を型パラメータとして使用できません。そのため、たとえば Stream<int> と記述することはできません。

ただし、前回の記事で、カワウソの平均年齢の計算部分（**リスト 3**）にそのようなコードがあると気づいた鋭い読者もいらっしゃるでしょう。

実際、このコードではパイプラインの大部分でプリミティブ型を使用しています。そこで、このコードを少し解剖し、プリミティブ型の使用方法について確認しましょう。

まず、double へのキャスト部分で困惑しないようにしてください。このキャストは、Java で整数の除算を実行するのではなく適切な平均を算出するためのものに過ぎません。

map() の引数はラムダ式になっており、このラムダ式で Otter を受け取って、int を返しています。 Java のジェネリクスを使用してこのコードを書き換えるとすれば、このラムダ式は Function<Otter, int> を実装するオブジェクトになります。しかし、Java のジェネリクスではプリミティブ型を含めることはできないので、戻り型が int であることを別の方法でコード化する必要があります。そのために、型の名前に戻り型を挿入します。

リスト1　リスト2　／リスト3

```
stream()   filter()   map()   collect()
Collection -> Stream -> Stream -> Stream -> Collection
```

すべてのリストのテキストをダウンロード

したがって、実際に推論される型は ToIntFunction<Otter> となります。 このような型は、関数型のプリミティブ型への特殊化と呼ばれます。プリミティブ型への特殊化は、int と Integer 間のボクシングとアンボクシングを避けるために使用します。 オブジェクトの不要な汎化が抑制され、使用しているプリミティブ型に固有の関数型を使用できます。

それでは、平均の計算をさらに詳しく見ていきます。 それぞれのカワウソの年齢を取得するために、以下の式を使用します。

```
ots.stream()
  .map(o -> o.getAge())
```

呼び出している map() メソッドの定義は以下のとおりです。

```
IntStream
map(ToIntFunction<? super T> f);
```

この定義から、特殊な関数型である ToIntFunction を使用していること、および Stream を特殊化した型を使用して int のストリームを表していることが分かります。

この後、処理を reduce() に渡していますが、reduce() の定義は以下のとおりです。

```
IntStream
map(ToIntFunction<? super T> f);
```

reduce() もやはり、int のみに対して動作する特殊化された型であり、2 つの引数を持つラムダ式（両方の引数が int）を受け取ってリデュース処理を実行しています。

reduce() は集合操作ですので、先行評価型のメソッドです。そのため、パイプラインは呼出しの時点で評価され、1 つの値を返します。返された値は double にキャストされ、全体の平均値に変換されます。

前回の記事でプリミティブ型に関する上記の詳細に気づかなかった方も、心配はいりません。 このような一部の差異を、ほとんど常に開発者から隠すことができる点が、型推論の優れているところの 1 つです。

これまで、Stream<T> という抽象概

念が Java 7 のコレクションよりも一般的である理由について説明しました。これ以降は、重要なユースケースである並列化について説明します。

並列操作

Project Lambda の主な目標の 1 つは、Java におけるコレクションのサポートをアップグレードして、マルチコア・プロセッサを効率的に利用できるようにすることです。

for ループは直列的です。 Java 7 以前のバージョンでは、コレクションに対するすべての操作も直列的です。 つまり、操作するコレクションの大きさを問わず、操作の実行に使用されるコアは 1 つしかありません。 これは、データ・セットが大きくなるほど不都合です。

ストリームの遅延評価アプローチによって、ラムダ式フレームワークで並列操作をサポートできるようになります。

Stream API の主な前提事項は、「Collection から作成する場合もその他の手法で作成する場合も、Stream の作成にはコストをかけるべきではない。ただしパイプライン内にコストのかかる操作が含まれる場合もある」というものです。 このような前提があるため、以下のような並列パイプラインを記述することができます。

```
s.stream()
 .parallel()
 .streamOps()
 .collect();
```

parallel() という 1 つのメソッドにより、それまで直列的であった Stream が並列化されます。 この並列化機能により、一般の開発者は parallel() を並列化のエントリ・ポイントとして使用できます。並列化サポートの提供はライブラリ作成者の担当となります。

最後に、Java プラットフォームでは Spliterator と呼ばれる新しい概念を使用して並列化を実装しています。その内部的な仕組みを確認しましょう。

Spliterator

並列化を行うために、まず 1 つの Stream を使用し、最終的に複数のストリームを使用します（さらに複数のストリームを最後に再結合する必要がある場合もあります）。 そのためには、反復処理および複数要素への分割が可能な抽象概念が必要になります。この抽象概念を、java.util.Spliterator という新しいインタフェース型で表しています。Spliterator には、tryAdvance() と trySplit() という 2 つのメソッドがあります。これらのメソッドはそれぞれ、Spliterator の反復処理と分割処理を表しています。

Stream を自動的に並列化できるようなライブラリの作成は簡単ではありません。そのため、個々の Stream に便利な特殊プロパティが用意されているのであれば、そういったプロパティを利用すべきです。 Spliterator インタフェースでは、便利な特性を一連のフラグとして宣言することで対応しています。 よく使用するフラグについて以下に示します。

- DISTINCT：ソースのすべての要素が一意であることを示す
- SORTED：要素がソート済みの順序で生成されることを示す
- SIZED：ストリーム内の要素数が有限かつ既知であることを示す
- NONNULL：ソースに null 要素がないことを示す
- IMMUTABLE：ソースの作成後、要素の順序を変更できないことを示す

Java 標準ライブラリは、さまざまなレベルで Spliterator をサポートします。

Spliterator を使用する簡単な方法は、Iterator を最初に取得し、java.util.Spliterators クラスのファクトリ・メソッドを使用して Spliterator を生成することです。 この方法で、java.util.stream.Streams のいずれかのファクトリ・メソッドを使用して Stream を容易に作成できます。

Spliterator には、他の Stream API と同様に、クラスのプリミティブ型への特殊化機能も含まれており、Java ジェネリクスの規則では許可されないような場所でプリミティブ型を使用できます。

遅延のパワー

Java 8 では、Stream での遅延評価の利用という、Java にとって新たなプログラミング・パラダイムが導入されようとしています。 遅延評価は極めて強力な新機能ですが、慣れるまでに少々時間がかかります。

まとめ

Java 8 では Stream API が導入されます。開発者はこの API を利用して、基本的な有限のコレクションよりも一般化されたデータソースを操作できます。 たとえば、Stream により無限集合を表すことができます。

Stream API では遅延評価が適用され、また、ラムダ式が大量に使用されています。そのため、Stream をどこまで評価するかをプラットフォームで完全に制御できます。 遅延評価およびラムダ式フレームワークにより、Java 7 以前のバージョンでは実現できなかった幅広い最適化への道が開かれます。たとえば、並列操作が可能になります。

自動並列化は、parallel() メソッドを使用することで実行できます。parallel() メソッドは、それまで直列的であった Stream を並列化するものです。 内部的には、Spliterator という新しい抽象概念が並列化機能の実装を担います。Spliterator には、他の Stream API と同様に、クラスのプリミティブ型への特殊化機能が含まれており、Java ジェネリクスの規則では許可されないような場所でプリミティブ型を使用できます。

この 3 回シリーズの記事では、Java 8 で導入されるラムダ式の機能について探究しました。 ラムダ式の構文を確認し、自動変換、型推論、コレクション・ライブラリのアップグレード（デフォルト・メソッドなど）について説明しました。 さらに、新しい Stream という抽象概念や遅延評価、プリミティブ型への特殊化についても説明しました。 最後に、自動並列化と、自動並列化の大部分を担う Spliterator と呼ばれるメカニズムについて紹介しました。

Java 8 のリリースによって、ラムダ式や関連のアップグレードが Java 開発者にとって非常に価値のあるツールとなることは確かです。 これらの機能の導入によって、Java プラットフォームの各機能が活性化され、ほぼすべての Java のコードベースおよびプロジェクトが大きな影響を受けることになるでしょう。 **</article>**

LEARN MORE

- ラムダ式に関する疑問がある場合や、世界中で開催されているHack Dayイベントに参加するためには、Adopt OpenJDKプロジェクトにご参加ください。

DIBY MALAKAR

BIO

Diby Malakar: Oracle Cloudの製品管理担当Director。

開発者をクラウドへと駆り立てるものとは

Java 開発者向けのクラウド入門書

年を追うごとに、私たちは普遍的につながった地球村という構想の実現に近づいています。 現在、インターネット・ユーザーの数は世界中で 20 億を上回っており、推計世界人口に対するインターネット普及率は、2012 年末の 34.3% から 2016 年末までには 45% に上昇すると予測されています。こうしたインターネット普及の広がりにより、ソフトウェア開発の分野においては、地理的に分散したチームやプロセスを利用することで、開発コストを削減し生産性と処理能力を向上させるまたとない機会がもたらされます。

2020 年に生成されるデータ量は 2009 年の 44 倍になると予想されており、世界全体の総計は 2015 年までに 7.9 ゼタバイト（8 兆ギガバイト）に達する見込みです。 開発者は、このデータを使用する、いっそう強力なアプリケーションを作成するよう迫られています。 また、企業はこの急激なデータ増加に対応するため、これまでにない能力、スケーラビリティ、スピードを備えたソリューションを探しています。

開発者に影響を与える傾向はもう 1 つあります。モビリティによって、個人が生産性を上げるために特定の場所にいる必要がなくなりました。 2011 年に販売されたスマートフォンは 4 億 2,000 万台、タブレットは 6,600 万台に上ります。 2016 年にはこれらの数字が膨れ上がり、12 億台のスマートフォンとタブレットが販売されると予測されています。 地球規模のプロジェクト・チームはコードを共有し、世界中どこにいても可能な限りシームレスに協業するようになります。 必要なのは、このような協業を実現するためのインフラストラクチャです。

全インターネット・ユーザーの 90%が少なくとも 1 つのソーシャル・サービス・アカウントを持ち、Facebook だけで 1,300 万を超える企業ページがある現在、ソーシャル・プラットフォームの導入は私たちの生活に欠かせない要素となりました。 ソーシャル活動が増すにつれ、現代の開発環境はこの増加に合わせて同レベルのシームレスで頻繁な対話を提供する必要に迫られています。

最先端インフラストラクチャの必要性から、企業は競争力を維持するための柔軟性、スケーラビリティ、応答性を実現するモダナイゼーションを求める圧力にさらされています。

しかし、インフラストラクチャ全体の再構築やスタッフの再教育に費やす余裕がある企業はめったにありません。標準ベースのクラウド・インフラストラクチャの導入が、企業の段階的なモダナイゼーションを可能にします。

このような傾向を受けて、アプリケーション・ニーズは急激に増加しています。 加えて、開発の速度と競争力によって生産性、スループット、品質に大きな圧力がかかっています。 しかし、開発リソースは依然として線形の増加を続けているため、" 開発ギャップ " が大きくなっています（**図 1** を参照）。 クラウドがこれほどまでに有益な開発戦力になったのは、この開発ギャップを埋める能力にあります。

Gartner によれば、パブリック・クラウド・サービス市場は、2013 年に世界中で合計 1,310 億ドルに達する見込みであり、2012 年の 1,110 億ドルから 18.5% の成長を遂げると予想されています。クラウド・サービスの市場規模と市場成長から見ると、市場力学は世界の地域別に大きく異なります。Gartner によると、アジア太平洋、中南米、東ヨーロッパ、中東、北アフリカの新興市場は全体としては小規模な市場ですが、もっとも高い成長率を示しています。 ただし、中国は例外であり、大規模で成長率も高い市場です。 北米、西ヨーロッパ、日本、成熟したアジア太平洋諸国といった成熟市場の規模は大きいながら、成長率は低くとどまっています。

クラウド・コンピューティングの基本モデル

クラウド・コンピューティング・プロバイダは、Infrastructure as a Service（IaaS）、Platform as a Service（PaaS）、Software as a Service（SaaS）という3つの基本モデルに基づいてサービスを提供しています。IaaSがもっとも基本的なモデルであり、それぞれの上位モデルは下位モデルの要素を抽象化したものになっています。 各モデルの特徴は次のとおりです。

IaaS：もっとも基本的なクラウド・サービス・モデルであり、IaaSプロバイダは物理マシンまたは（たいていは）仮想マシンの形をとったコンピュータとその他のリソースを提供します。

アプリケーションをデプロイするには、クラウド・ユーザーがオペレーティング・システム・イメージやアプリケーション・ソフトウェアをクラウド・インフラストラクチャにインストールします。 このモデルでは、クラウド・ユーザーがオペレーティング・システムおよびアプリケーション・ソフトウェアに対するパッチ適用と保守を行います。

PaaS：PaaSモデルでは、通常、クラウド・プロバイダが提供するコンピューティング・プラットフォームにオペレーティング・システム、プログラミング言語の実行環境、データベース、Webサーバーが含まれています。 アプリケーション開発者は、基盤となるハードウェアおよびソフトウェア・レイヤーを購入して管理するためのコストと複雑さを伴うことなく、このようなクラウド・プラットフォーム上でソフトウェア・ソリューションを開発し、実行できます。

SaaS：SaaSモデルでは、クラウド・プロバイダがクラウドにアプリケーション・ソフトウェアをインストールして運用し、クラウド・ユーザーはクラウド・クライアントからソフトウェアにアクセスします。 アプリケーションが稼働するプラットフォームやクラウド・インフラストラクチャの管理をクラウド・ユーザーが行うことはありません。 このサービス形態により、クラウド・ユーザー自身のコンピュータにアプリケーションをインストールして実行する必要がなくなるため、保守とサポートが容易になります。

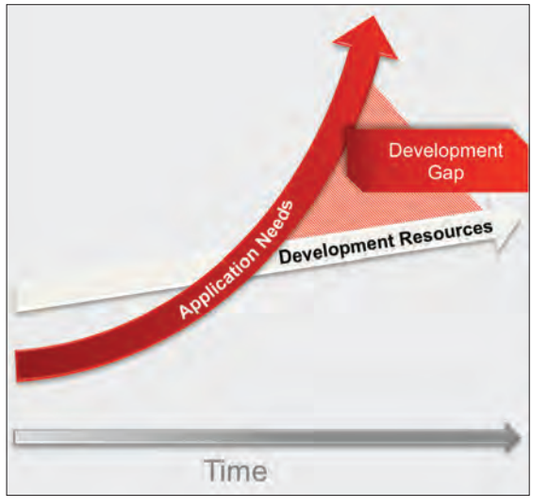


図1

IaaSクラウド・ソリューションを利用すると、最小限の労力でコンピューティング、ネットワーク、ストレージといったリソースをデプロイできます。 このような機能は開発者から見て非常に便利なものですが、クラウドの利点はこれまで述べたものにとどまりません。 実際、インフラストラクチャ・レイヤーのプロビジョニング以外の部分に開発者は80%を超える時間と労力をかけており、ここに、より高い価値があります。

PaaSはこの部分で、アプリケーション

スポットライト：Oracle Java Cloud Service

Oracle Java Cloud Serviceは、クラウドでビジネス・アプリケーションを開発してデプロイするためのエンタープライズ・クラスのプラットフォームを提供します。 このサービスはクラウド内のOracle WebLogic Serverを使用して、選択の自由、オープン環境、セキュリティ、および使いやすさを提供します。

Oracle Java Cloud ServiceはJava標準をサポートしており、柔軟性、移植性、デプロイに関するオプションを提供します。

選択の自由：開発者は、IDE、アプリケーションの開発とデプロイ、サービス統合、およびサービスの監視と管理に関して、自由に選択できます。 IDEについては、EclipseやOracle JDeveloperからNetBeansまでの各種プラットフォームがサポートされており、ボタンをクリックするだけで、Oracle Java Cloud Service上にアプリケーションをデプロイできます。 開発サイクルにチーム開発が必要な場合、主要な開発環境からOracle Java Cloud Serviceまたはその他の標準ベース環境へと、ご使用のアプリケーションを簡単にデプロイできます。 また、サービス統合に関してもWebサービス、SOAP、REST APIなどの選択肢があります。 サービスの管理と監視については、MavenおよびAntのプラグインを備えたコマンドライン・インタフェースがあります。

オープン環境：Oracle Cloud内でJava EEアプリケーションを作成し、標準のWAR（Web Application Archive）またはEAR（Enterprise Archive）形式でデプロイできます。Oracle Java Cloud Serviceは標準のJava EE仕様だけでなく、Oracle WebLogic Server固有の拡張機能やOracle Application Development Frameworkのコンポーネントを使用したアプリケーションのデプロイをサポートしています。

セキュリティ：Oracle Java Cloud Serviceではインスタンスが分離されているため、複数のレイヤーでアプリケーションが保護されます。 サービスには専用の仮想マシン（VM）とスキーマが含まれており、Oracle Exalogic Elastic Cloudの上層に配置されています。VMは、Oracle Exadata Database Machine上の該当するデータベース・スキーマにマップされます。 セキュリティが確保されるのは、Oracle Java Cloud Serviceにデプロイされたアプリケーションだけではありません。 Oracle Fusion Applicationsなどの別サービスをOracle Cloud内で使用しており、Oracle Java Cloud Service内にカスタム拡張機能を構築している場合、Oracle Cloudの共通インフラストラクチャ内のIDサービスを使用して、Oracle Fusion ApplicationsからOracle Java Cloud Serviceへと、セキュアな方法でID情報とユーザー・プロファイルを取り込むことができます。

使いやすさ：サービスの加入から監視や管理までのサービス・ライフサイクル全体の処理を、使いやすい1つのWebベース・インタフェースから実行できます。

この他に、オラクルは、Oracle Database Cloud Service、Oracle Developer Cloud Service、Oracle Storage Cloud Service、Oracle Messaging Cloud Serviceなどの、リッチ・アプリケーションを構築するための包括的なプラットフォーム・サービスのセットを提供しています。

の開発、テスト、デプロイ、管理に対して非常に大きなメリットを提供します。

オンデマンド・インフラストラクチャ、自動プロビジョニング、柔軟な開発 / ランタイム・ツール、および管理の自動化を備えた IaaS および PaaS のクラウド・サービスは、開発者エクスペリエンスに重要な影響を及ぼすでしょう。

ギャップに注意

アプリケーション・ニーズは急激に増加していますが、開発リソースは依然として線形の増加を続けているため、"開発ギャップ" が大きくなっています。

クラウド製品に求めるもの

開発者向けのクラウド・オプションを評価する場合、標準、選択肢、幅広さ、奥深さ、セルフサービス機能、データの整合性と移植性を考慮に入れる必要があります。

標準のサポート：対象のクラウド・サービスは、SQL や Java、HTML5 などの標準に基づいて構築されていますか。 標準ベースの構築には多数の利点があります。 もっとも直接的な利点の 1 つは、クラウドでアプリケーションをより透過的に実行できる点です。つまり、コードを何ら変更することなく、ローカルのオンプレミス環境にアプリケーションを移植できます。 反対に、オープン標準に基づいて構築されたオンプレミス・アプリケーションを実行しており、そのアプリケーションをクラウドに移行したい場合は、変更実施プロセスに沿って処理を進めます。

選択肢：パブリック・クラウドとプライベート・クラウドだけでなく、該当する場合はベンダーの製品ポートフォリオを含んだクラウド・サービスが選択できることが望ましいです。 ベンダーが提供する製品は、クラウド・プロバイダがプライベート・クラウドを構築して管理する、包括的な製品ですか。また、顧客がクラウドを完全に管理 / ホストするためのマネージド・クラウド・サービスは提供されていますか。 プライベート・クラウドを利用すると、組織はセキュリティ、規制遵守、サービス・レベル、機能を完全に制御し、可視化することができます。

幅広さ：クラウド・コンピューティングによるもっとも大きな利点の 1 つは、組織全体にわたるサイロが解消されることです。 クラウド・ソリューションは、サイロ化した複数のパブリック・クラウドを使用した場合に発生するデータやビジネス・プロセスの断片化を防ぎます。 さまざまな種類のクラウド・サービスがベンダーから提供されている場合は、最大の柔軟性が得られます プラットフォーム・サービスでは、開発者コミュニティによるアプリケーションの構築とデプロイが可能でなければなりません。 アプリケーション・サービスは、パッケージ化された機能をビジネス・ユーザーがただちに使用開始できるものである必要があります。

たとえば、ERP やカスタマ・リレーションシップ・マネジメント、人材管理などのパッケージ・アプリケーションがサブスクリプション・ベースで提供されていますか。 ソーシャル・サービスがアプリケーションに統合または追加されていますか。

奥深さ：機能性に優れているだけでなく、より重要な点として、シームレスに連携するように統合されているサービスを探します。 サービスは実績あるインフラストラクチャ上で稼働しており、エンタープライズ・レベルのソリューションが持つ信頼性とパフォーマンスを提供していなければなりません。

セルフサービス機能：ベンダーや IT スタッフの関与を最小限に抑えて、アプリケーションやプラットフォームの開発、デプロイ、管理を実行できますか。

データの整合性と移植性：ベンダーは複数の顧客のデータを混合していますか。 意外なことに、こうしたデータの混合は、パブリック・クラウド環境では非常によくあることです。 ベンダーがデータの独立性を確保できるかどうかを確認すると良いでしょう。 また、データの移植性はどうでしょうか。 ベンダーによってデータの囲い込みが行われないよう注意しましょう。 簡単かつ分かりやすい方法で、すべてのデータを使用可能な形式でクラウドから取得できることを確認します。

まとめ

開発者はクラウドの利用を通じて、アプリケーションをより迅速に構築してデプロイし、イノベーション・コストを削減し、リスクを大幅に軽減するよう迫る圧力によって生じた "開発ギャップ" に対応することが可能です。 優れたクラウド・ベンダーは、オープン標準に基づき、シームレスに連携するよう統合された包括的なテクノロジーおよびアプリケーション・スイートを提供しています。 **</article>**

LEARN MORE

- Oracle Java Cloud Serviceチュートリアル
- 『Deploying an Application to Oracle Java Cloud Service』チュートリアル
- 『Securing a Web Application on Oracle Java Cloud Service』チュートリアル
- Oracle Java Cloud Serviceの30日間無料試用

JAVA EE 7 の到来

Anil GaurがJava EEの進化を語る

STEVE MELOAN

Java EE 7プラットフォーム・スペック・リード、Bill ShannonおよびLinda DeMichielと話すオラクルのソフトウェア開発担当Vice President、Anil Gaur（中央）

写真：PHIL SALTONSTALL

エンタープライズ Java コンピューティングの業界標準として、Java Platform, Enterprise Edition はエンタープライズ分野の新たな使用法、パターン、フレームワーク、テクノロジーに合わせて絶え間ない進化を続けてきました。 そしてついに、HTML5、WebSocket、Java API for RESTful Web Services（JAX-RS）、JSON Processing（JSON-P）など、最近の Web アプリケーションに必要不可欠なサポートを備えた、Java EE 7 がリリースされました。

Java Magazine はオラクルのソフトウェア開発担当 Vice President である Anil Gaur とのインタビューで、リリースされたばかりの Java EE 7 プラットフォームに含まれる新たな機能およびテクノロ

Java EE 7の機能について、オラクルのJava Evangelist、Arun Gupta（濃紺のシャツ）およびオラクルのEngineering Manager、Shreedhar Ganapathyと話し合うGaur

ジーについて取り上げたほか、Java EE 7 がエンタープライズの開発者と顧客に与える影響について話をうかがいました。

Java Magazine： Java EE 7 において注目すべき新機能を教えてください。

Gaur： Java EE プラットフォームは現在、非常に重要な時期にあります。新たな Java の使用法やパターンに対応していくために、私たちは迅速に前進する必要があります。 このような変化の大部分は最近の Web アプリケーションによってもたらされています。 過去数年の間に他の多数のフレームワークが発展を遂げました。私たちも、これらの変化に合わせてプラットフォームを提供したかったのです。

これを踏まえ、Java EE 7 の主要テーマは生産性と HTML5 のサポートです。 HTML5 の機能は、WebSocket、Java Servlet 3.1、NIO（ノンブロッキング I/O）機能、Server-Sent Events（SSE）、JAX-RS、JSON-P によって強化されています。

また、プラットフォーム全体を通じて生産性を向上させるための機能強化がいくつか実施されています。というのは、生産性向上はもう 1 つの課題であり、開発コストやアプリケーション保守に直接的な影響を与えるためです。 Java EE 7 では、開発者が書く必要のある定型コードは少なくなっており、プラットフォームでもバッチ処理や同時実行性などの各種の新しい仕様を通じて、より豊富な機能を提供しています。

さらに、JMS（Java Message Service）の仕様が JMS 2.0 で大きく改善されており、Servlet 仕様や EJB（Enterprise JavaBeans）仕様において以前から提供されている概念がいくつか導入されています。 Expression Language(式言語) の仕様も Expression Language 3.0 で機能強化されているため、プラットフォームに含まれるその他多数のテクノロジーに利点がもたらされるでしょう。

変更された各種仕様は、JPA 2.1（Java Persistence API）、Java Servlet 3.1、EJB 3.2、Bean Validation 1.1、および CDI（Contexts and Dependency Injection for the Java EE platform）です。CDI は Java EE 6 で導入されており、もっとも広く使用されている仕様の 1 つです。

Java Magazine： Java EE 7 では、開発の簡素化とアプリケーションの高機能化をどのように両立させていますか。

Gaur： プロファイル概念の導入に関しては、簡素化を特に重視しました。 プラットフォームが巨大になったため、的を絞った限定的な使用には適用しづらい場合があると何度も聞いており、この報告を非常に深刻に受け止めていました。 このため、Java EE 6 プラットフォームには多大な労力を投じて、Web アプリケーション開発を特に対象とした標準 API のランタイム・スタックである Web Profile を導入しました。 また、このプロファイルのおかげで、今後のプロファイルの定義に使用できる基本的なフレームワークを導入できるようになりました。また、デフォルト設定を増やし、さらに CDI を活用したことで、XML コーディングの使用量が減ったため、ここでも開発の簡素化が実現しています。

Java EE 7 で追加した新しい仕様に対する要望は大きいものでした。WebSocket や JSON などのテクノロジーはかねてから使用されていましたが、Java EE 7 のリリース前は標準 API が存在しなかったのです。 もう開発者が独自の JSON ライブラリをアプリケーションに組み込む必要はありません。サポートはベース・プラットフォームに含まれています。 さらに、WebSocket のエンドポイントを Java クライアントから直接呼び出せるようになったため、開発が大幅に簡素化されると同時に、デプロイするアプリケーションのサイズも縮小されます。

その他にプラットフォームに追加された

バッチ処理に対しては、顧客から定期的に要求が寄せられていました。 この追加により、バッチ・アプリケーションを実装するための標準化されたプログラミング・モデルが提供されます。JSR ではドメイン領域、バッチ・ジョブ、アプリケーション・エグゼキュータ、ジョブ・マネージャが定義されています。 Java EE でのサポートに加えて、一部の機能は Java SE でも提供される予定ですが、Java EE は Java SE のスーパーセットであるため、本質的により多くの機能セットが含まれています。 例を挙げると、JTA（Java Transaction API）は Java EE のみに含まれます。

また、今回プラットフォームに追加された同時実行性も、開発者から同様に要求されていたものです。 同時実行性により、異なる実行スレッドを使用することで、アプリケーション・レベルの処理を同時に実行できるようになります。 以前のリリースでは、さまざまなアプリケーション・サーバーごとに独自仕様の実装が加えられていましたが、この場合、アプリケーションの移植性が損なわれます。Java EE 7 に同時実行性が追加されたことで、同じ標準 API をすべての EE ベンダー、顧客、開発者が利用できるようになりました。

その他に、前述したとおり、コミュニティからリクエストされていたさまざまな点で JMS 2.0 仕様が改訂され、簡素化が実現しています。 JMS 2.0 によって、前のバージョンのプラットフォームでの EJB 仕様や Servlet 仕様と同様に、アノテーションがより多く活用されるようになるでしょう。

JMS コネクション・ファクトリもまた、Dependency Injection（DI）を利用します。さらに、中間オブジェクトが不要になるように API が整理されており、AutoCloseable リソースも提供されています。 これらの変更は開発者にとって非常に魅力的なものであり、開発者は JMS をより積極的に活用するようになるでしょう。

確かにプラットフォームには API が追加されましたが、その大半はコミュニティからの要望に基づくものか、エンタープライズ・アプリケーション開発における問題解決に必要とされるものです。

Java Magazine：このリリースで排除された機能は何でしょうか。また、機能を排除することによる効果やアプリケーションの下位互換性への潜在的な影響についてお聞かせください。

フィードバックに応える

プラットフォームが大きくなりすぎて融通がきかないため、的を絞った限定的な使用には適用しづらい場合があると何度も聞いており、この報告を非常に深刻に受け止めていました。

Gaur：Java EE 7 には多数の新機能が追加されていますが、オプションになった API もあります。 プラットフォームは機能面では強化されていますが、一部の古い機能のクリーンアップも実施されているため、プラットフォームのフットプリントが縮小されるだけでなく、開発者が古い仕様に対処する必要がなくなります。

この手続きが開始された Java EE 6 では、いくつかのテクノロジー（Java EE の管理（JSR 77）、アプリケーション・デプロイメント（JSR 88）、UDDI レジストリとのインタフェースをとる JAXR（JSR 93）、XML ベースの RPC 向けの JAX-RPC（JSR 101）、EJB 2.x の Container Managed Persistence（実質的に Java Persistence API（JSR 317）で置換））がオプションになり、最終的な排除の対象になりました。

これらのテクノロジーは Java EE 6 リリースで排除の対象となっていたので、Java EE 7 リリースで排除することも可能でしたが、排除するかどうかは任意で選択できます。 その決断は、それぞれのベンダーに任されています。引き続き需要があったり、これらの仕様を使用する顧客基盤があったりする場合は、ベンダーはプラットフォームに仕様を保持できます。 ただし、次のリリース（Java EE 8）でこれらの仕様は排除する予定です。 ほとんどのベンダーがこの戦略に従うものと思われます。

Java Magazine：Java EE 7 の Web Profile スタックがこのリリースにもたらすもの、また Web アプリケーション開発者に及ぼす影響についてうかがえますか。

Gaur：Web Profile は Java EE 6 で導入され、非常によく使用されるようになりました。Web Profile はフル・プラットフォームのサブセットとして、大半の Web アプリケーションのニーズを満たすことを対象とした標準 API を提供します。 すべての開発者がすべての Java EE API を必要とするわけではありません。Web Profile を使用すれば、ランタイムのメモリ・フットプリントが縮小され、インストールが簡単になり、応答性が向上します。

Java EE 6 と同様に、Java EE 7 リリースにも Web Profile が含まれていますが、WebSocket や JSON-P などの仕様が新たに追加されています。また、JAX-RS 2.0 の拡張によるクライアント API の追加も注目に値します。 もちろん、これらの仕様は Web Profile だけに限ったものではありません。

アプリケーションに JavaScript や HTML5 を使用する開発者はますます増えており、

陽気なひとときを過ごすGaurとJava EE 7チーム

WebSocket および JSON-P はこのプログラミング・モデルとの整合性が非常に高い仕様であることから、Web Profile は次世代の Web アプリケーション開発に適しており非常に魅力的です。

Java Magazine： 当初は Java EE 7 を対象としていたクラウド関連機能が Java EE 8 に延期された決定について、また関連する機能についてお聞かせください。

Gaur： 難しい決断でしたが、正しい判断だったと考えています。 最終決断の前には、コミュニティ・メンバーや専門家グループ、そしてベンダーとも協議しました。

この決定の一因は、クラウド分野、特にプロビジョニングやマルチテナント、伸縮性の成熟度が不十分であることにありました。 私たちがこの事実を認識したときは、すでに実装がかなり進んでいました。 結局のところ私たちは、十分に目的を果たさない、あるいは目的に合わない機能や API はを追加することは避けたいと考え、業界や開発者に最大の利益がもたらされる方法で目標を確実に達成するように、より慎重な方法を選ぶことにしました。 コミュニティ・メンバーと意見交換し、アンケートを実施したのち、Java EE 7 の焦点を HTML5 および生産性に移すことに決めました。

このように範囲を変更したことで、機能強化や簡素化、ユーザビリティへの注力を続けながら、スケジュールどおりにリリースすることができたのです。 つまるところ、完全には実現しないもののために、すでに準備が整った機能を延期するようなことはしたくなかったのです。

Java Magazine： 延期された JSR を具体的に教えてください。

Gaur： そのような特定の JSR はありません。クラウド要件は分野横断的であり、さまざまな仕様に影響を与えています。 したがって、プロセス内で実際に削除された仕様はありませんが、Java EE 7 プラットフォーム仕様を含む、いくつかの仕様に含まれる機能が影響を受けています。

Java Magazine： 当面の間、クラウドで Java EE アプリケーションを実行するためにどのようなサービスを利用できますか。

Gaur： 私たちにとって喜ばしいことに、世の中に数多くあるベンダーに目を向けてみると、ほとんどのクラウド・プラットフォームが Java で構築されています。 そして、Java EE 6 はすでにこれらの環境に対応しています。 したがって、Java EE が持つコンテナ・ベース・モデルにセキュリティの分離を組み合わせると、PaaS（Platform as a Service）対応アプリケーションの作成に適したプラットフォームになります。 また、API for RESTful Web Services（JAX-RS）もクラウド・アプリケーションに合うよう十分に調整されています。 これらを踏まえると、クラウド・ソリューションの構築に適したプラットフォームはやはり Java EE であると言えます。

Java Magazine： Java EE 7 の発展と改善に対して、Java コミュニティはどのように関わってきたのでしょうか。 特に最近のクラウド関連の決定を中心にお聞かせくださいますか。

Gaur： Java EE 5 以降、Project GlassFish のもとでオープンソース・プロジェクトとして Java EE のリファレンス参照が開発されてきました。 Java EE 7 ではここから一段と踏み込んで、すべての Java EE 仕様が専門家グループや Java EE ベンダーだけでなく、より幅広いコミュニティが参加して開発されるようになりました。 したがって、この 2 ～ 3 年で仕様開発プロセスの透明性がますます高くなっているのは、偶然ではありません。 オラクルとコミュニティは、より多くのコミュニティ・メンバーがより上流のプロセスから関与することで、コミュニティが仕様の開発に対してレビューを行い、フィードバックを提供できるようにすることを強く望んできました。

私たちは、透明性のある方法で JSR を運用するためのルールを定義する目的で JSR 348 を登録しました。 現在、オラクルがリードするJSRはすべてこのプロセスに従っています。 専門家グループは技術的なディスカッション向けにオープンな電子メール・エイリアスを提供しており、仕様の暫定ドラフトはプロジェクト・ページ上で参照可能であり、各 JSR の対象機能の一覧を含む JIRA が使用されています。 つまり、興味がある人ならだれでも JIRA リクエストを登録して、特定の機能の追加を求めたり、計画対象となった機能に関するフィードバックを提供したりすることができます。 それぞれの JSR に用意された JIRA リクエスト用のダッシュボードにより、

はるかに幅広いコミュニティから継続的なフィードバックが提供されるようになりまし

オラクルのサンタクララ事業所（カリフォルニア州）でチーム・メンバーのRajiv Mordani（左）およびNazrul Islamと話すGaur

た。 その結果、より速いペースでより優れたテクノロジーを提供することができます。 さらに、19 の Java ユーザー・グループ（JUG）が Adopt-a-JSR と呼ばれるプログラムに参加してくれました。このプログラムでは、開発者の有志が、提案された内容をレビューし、プラットフォーム上のテクノロジーへの潜在的な影響を調査します。 また、JUG は提案内容を試すため、いくつかの小さなアプリケーションの開発も行いました。 これらの JUG から寄せられたフィードバックによって、プラットフォームが完成するずっと前に問題が解決されました。

Java Magazine： 現時点で Java EE 8 に予定されている機能について、また目標とするリリース日についてうかがえますか。

Gaur： Java EE 7 のリリース後、オラクルとその他のベンダーが Java EE 8 への取組みを開始する予定です。このリリースの重点は、Java EE 7 で延期された標準ベースのクラウド・プログラミング・モデルに置かれ、マルチテナントと伸縮性のサポートが追加される予定です。

また、Java EE 8 では Jigsaw ベースのモジュール化も提供される予定です。Java EE プラットフォームで何らかの NoSQL をサポートすることにも強い関心が寄せられているため、NoSQL サポートの評価も行われます。その間も、プラットフォームの進化は続きます。 Java EE 7 より後の新機能追加が Java EE 8 に限られるわけではありません。JCache や JSON-B（JSON Binding）などの新しい JSR は Java EE 8 よりずっと前に完成させたいと考えています。つまり、Java EE 8 が最終的にリリースされる前に、顧客が新たな API を確認でき、試せるように、プラットフォームは段階的な発展を続けていきます。

Java EE 8 の正式なリリース日に関してですが、リリースのスコープおよび機能の定義をコミュニティが支援する予定であり、その定義によって最終的に日付が決定されます。 最終的な機能セットとリリース日の策定については、専門家グループが中心的な役割を担うことになります。

Java Magazine： JavaOne 2012 で Hasan Rizvi が指摘したとおり、現在の Java の成功は、技術革新、コミュニティの参加、オラクルの管理責任という三者関係に基づいています。 どのようにすれば、オラクルと Java コミュニティとの間のパートナーシップをさらに強化できるとお考えですか。

Gaur: オラクルの管理下で、Java コミュニティとの関係は大幅に強化されたと自負しています。 この事実はさまざまな面で実証されてきました。 コミュニティは次世代プラットフォームの定義に非常に積極的に関与しており、今後もこの積極的関与が続くものと期待しています。また、私たちは引き続きオープン形式で専門家グループを運営していく予定です。 JCP への参加や大々的な手続きの必要はなく、より多くの人々の参画が可能になります。また、オープンソースとしてのリファレンス実装の開発を継続することで、素早くフィードバックを集められるため、最終的には企業顧客と開発者の双方に訴えるプラットフォームの開発が可能になると考えています。

このオープン手法の証しとして、Java EE 6 リリースはこれまでのどのリリースよりも迅速に取り入れられました。 この迅速さは、より早い段階でのコミュニティの関与を実現したことに起因すると考えており、これからもこの方針により同様の肯定的な結果が得られることを期待しています。

Java Magazine： 開発者が Java EE 7 にアクセスする方法を教えてください。

Gaur： もっとも一般的なバンドルである Java EE SDK を使用して、開発者はプラットフォーム上の新技術を習得できます。 Java EE SDK は現在、Oracle Technology Network からダウンロードできます。 SDK バンドルは、Java

EE のリファレンス実装でもある GlassFish Server Open Source Edition 4.0 で構成されており、新しい API を扱った包括的なチュートリアルや多数のサンプル・プログラムが含まれています。これらのサンプル・プログラムを利用することで、開発者は各仕様を理解し、素早く習熟することができます。

IDE のサポートについても触れておきましょう。 NetBeans IDE は Java EE 7 の API をサポートする最初の IDE になります。 私たちはプラットフォーム開発にあたって NetBeans チームと非常に密接に協力することで、NetBeans IDE と GlassFish との間に極めて緊密な統合を実現するとともに、追加設定なしでプラットフォームをすぐに使用できるようにしました。 また、次に Java EE 7 プラットフォームをサポートする IDE としては、Eclipse と IntelliJ が見込まれます。

Java Magazine： Java EE 7 と併せてリリースされる GlassFish 4.0 はどのようなものになりますか。

Gaur： Java EE 5 および Java EE 6 と同様に、Java EE 7 のリファレンス実装は Project GlassFish から派生しています。 リファレンス実装として、GlassFish には常に最新の Java EE 仕様が反映されています。 したがって、Java EE 7 プラットフォームの出荷時に、GlassFish コミュニティは GlassFish Server Open Source Edition 4.0 を手にすることになります。 GlassFish Server Open Source Edition 4.0 は Java EE 7 の API すべてをサポートしていますが、クラスタ化機能と高可用性機能は評価目的でのみ提供されています。 本番デプロイを主体としたこれらの機能は、後続リリースである 4.1 で利用可能になります。

GlassFish 4.0 のダウンロードには 2 つの種類があります。 1 番目のディストリビューションには Java EE 7 のすべての API に対するサポートが含まれていますが、2 番目に含まれるのは Java EE 7 の Web Profile 仕様で定義された、API のサブセットのみです。 したがって、開発者は GlassFish 4.0 のオープンソース部分を含む SDK バンドルから始めることも、Java.net から個別に GlassFish 部分をダウンロードすることもできます。 **</article>**

コミュニティの声を聞く

確かにプラットフォームには API が追加されましたが、その大半はコミュニティからの要望に基づくものか、エンタープライズ・アプリケーション開発における問題解決に必要とされるものです。

Steve Meloan：元C/UNIXソフトウェア開発者。Wired、Rolling Stone、Playboy、SF Weekly、San Francisco Examinerなどの各誌にWebやインターネットに関する記事を執筆している。 近著はサイエンス・アドベンチャー小説の『The Shroud』。また、The Huffington Postに定期的に寄稿している。

LEARN MORE

- Java EE 7のダウンロード
- GlassFish 4.0
- NetBeans IDE

JOHAN VOS

BIO

Java EE 7で実現される生産性向上について、著者**Johan Vos**の解説をお聞きください。

Johan Vos：1995年以来 Java に従事。Linux への Java 移植を支援した Blackdown チームのメンバー。ソーシャル・ネットワーキング・ソフトウェア向けの Java ベース・ソリューションに取り組む LodgON の共同創設者。組込み開発とエンタープライズ開発の両方に熱心であり、徹底して Java に専念。詳しくは、ブログを参照。

写真：
TON HENDRIKS

Java EE 7

Java EE 7による開発生産性の向上

Java EE 7の仕様拡張と新規APIを利用して生産性を向上する

Java EE 7 は、Java EE 6 を徹底的に見直した待望のバージョンです。 Java EE はリリースのたびに新機能が追加され、既存の仕様が強化されます。 Java EE 7 は Java EE 6 の成功を土台に、引き続き開発生産性の向上を重視しています。

Java EE は、多くの仕様を包括しています。 JSR 342 は、コンテナの要件を網羅した仕様であり、Java EE プラットフォームでサポートすべきテクノロジーを含んでいます。Java EE Platform Specification プロジェクトには、専用の wiki もあります。 Java EE の仕様書には以下のように、開発の容易さ（Ease Of Development）が Java EE 7 の主要目標の 1 つであると明記されています。

「Java EE 7 でも、Java EE 5 と Java EE 6 の『開発の容易さ』という重点項目を踏襲します。 特に、Java EE 7 では JMS 2.0 API を改訂し、大幅に簡素化しています。 開発の容易さには設定の容易さも含まれます。」

Java EE 仕様の初期バージョンは、開発者から非常に複雑で扱いづらいという批判を多く受けました。「複雑」か「単純」かで言えば、「複雑」の方向に進んでいたのは明らかでした。

Java EE 5 より、開発の容易さに重点が置かれるようになりました。

アノテーションの利用、およびデフォルト動作とデフォルト設定の提供によって、コードや XML 設定の行数を大きく削減することが可能になりました。

Java EE 5 の簡素化に関する要件は以下の 2 つで構成されます。

- 大部分のアプリケーションでデフォルト値を適用できるようにし、かつ、デフォルト値をほとんどの開発者にとって使いやすいものにすること
- アノテーションやディスクリプタの提供により、デフォルト値を常に変更できるようにすること

Java EE 仕様はこれらの要件を満たすことで、「設定より規約（Convention over Configuration）」という設計パラダイムを忠実に守りました。 すなわち、規約に従う動作についてはコードや XML を追加する必要はなく、規約に従わない動作が求められる場合は、コードを記述するか XML を宣言することで実現できるようにしました。 その結果、Java EE での開発への障壁が明らかに低くなりました。

Java EE 6 仕様でもこのパラダイムは引き継がれ、着実に進展して、Java REST API を定義する JAX-RS などの多くの新規 API が追加されました。

Java EE 7 では、デフォルトの開発と設定がさらに簡素化されます。 Java EE 6 の既存のテクノロジーに対する変更点は多数ありますが、本記事では、開発生産性の向上につながる以下のような既存仕様の変更点を取り上げます。

- JPA 2.1（JSR 338）：スキーマ生成の標準化。 JPA を使用して表、索引、ジェネレータなどを作成可能
- JPA 2.1：ストアド・プロシージャも新たにサポート
- JPA 2.1：コンテナでのデフォルトのデータソース提供を必須化
- Servlets 3.1（JSR 340）：NIO2 ベースの非同期 I/O が可能
- JavaServer Faces（JSF）2.2（JSR 344）：HTML5 を新たにサポート

- JSF 2.2：ファイル・アップロード・コンポーネントの追加
- Bean Validation 1.1（JSR 349）：メソッドレベルの制約に対応
- Contexts and Dependency

さらに簡素に

Java EE 7 では、デフォルトの開発と設定が**さらに簡素化**されます。

Injection（CDI）1.1（JSR 346）：開発者がインターセプタの順序を指定可能。 また、@Vetoed アノテーションを使用することで、型またはパッケージを CDI の対象から外すことが可能

- Expression Language（EL）3.0（JSR 341）：ラムダ式のサポート

しかし、Java EE 7 は単なるメンテナンス・リリースではありません。 以下のような新規 API も追加されています。

- JSR 352：バッチ処理
- JSR 353：Java API for JSON Processing
- JSR 356：WebSocket
- JSR 236：並行処理

これらの新規 JSR が Java EE プラットフォームに追加されたのは、いずれも今日の Java EE 開発者にとって意義があるためです。エンタープライズ・コンピューティングの世界は常に進化しており、Java EE 標準も同様に進化しなければなりません。 もっと具体的に言えば、Java EE 仕様は、企業の開発者が、定型的なコードを大量に記述することも特定のベンダーへの依存に陥ることもなく、最新の優れたテクノロジーを利用できるような仕様であることが求められます。そのために、標準化が必要となるのです。

新規の JSR は例外なく、開発者の真のニーズに焦点を当てています。これにより、開発生産性が向上し、開発者は核となる業務に注力できるようになります。

バッチ処理

実世界のビジネス・アプリケーションの多くは、何らかのバッチ処理の手段を必要としています。 時々あるいは定期的に、非対話型のデータ処理を長時間行う必要が生じるものです。 典型的な例としては、1 日に 1 回、在庫を計算してレポートを生成するようなジョブが挙げられます。

このようなバッチ処理はすでに Java EE で実現できてはいましたが、開発者が独自のコードを記述するか、既存の（標準化されていない）サードパーティ製フレームワークを利用する必要がありました。 しかし、JSR 352 により、バッチ処理に関連するエンティティと、エンティティ間の関係が定義されました。 既存のバッチ・フレームワークの利用者にとっては非常になじみ深い用語や概念が導入されています。

Java EE 開発者は JSR 352 を利用することで、ジョブを XML で定義できます。ジョブはステップで構成され、ステップはチャンクで構成されます。その構成方法を**リスト 1** に示します。

ジョブ実行の典型は、データの読取り、データの処理、結果の書込みで構成される一連のプロセスです。 Java クラスは、データの読取り、処理、書込みに使用されます。 **リスト 1** の XML の例には 3 つの Java クラスが含まれます。この 3 つのクラスを**リスト 2** に示します。

JSR 352 を利用すれば、バッチ処理に必要となるインフラストラクチャについて開発者が考慮する必要はなくなり、ドメイン固有の問題に集中できるようになります。

リスト1　リスト2

```
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee">
  <step id="myStep" >
    <chunk item-count="3">
      <reader ref="myItemReader"></reader>
      <processor ref="myItemProcessor"></processor>
      <writer ref="myItemWriter"></writer>
    </chunk>
  </step>
</job>
```

すべてのリストのテキストをダウンロード

JSON

多くの Web サービスで、JSON 形式を使用してリクエストの受信とレスポンスの送信を行っています。 また、非常に多くの Web フレームワーク（Java、Java 以外の両方）が JSON 形式を採用しています。 そのため、多くの Java EE アプリケーションで JSON コードの解析や生成の機能が必要です。 さまざまな JSON パーサー / ジェネレータが提供されてはいますが、そういったパーサー / ジェネレータを使用すれば、非標準のインタフェースに強く依存することになります。

JSR 353 では JSON の解析に関する概念を標準化しています。 JSR 353 API を利用することで、独自仕様の API ではなく、javax.json パッケージ（およびそのサブパッケージの javax.json.spi と javax.json.stream）内の標準 API を利用できるようになります。

下層の実装は開発者から隠されており、JSON パーサーの提供者がこの標準 API を実装することを想定しています。

JSON 処理用の API は、XML 処理用の API に似ているところがあります。 多くのアプリケーションでは XML と JSON の両方を扱う必要があるため、このような類似性は重要です。 両方の形式に対して同じようなアプローチをとることができるため、開発が容易になります。

XML の処理と同様に、JSON データの処理には 2 つの方法があります。 1 つはストリームベースの処理、もう 1 つはドキュメントベースの処理です。

ストリームベースのアプローチでは、イベントごとに JSON ドキュメントを読み込んで処理します（**リスト 3**）。 一方、ドキュメントベースのアプローチでは、JSON ドキュメント全体を読み込みます（**リスト 4**）。これは、Document Object Model（DOM）を使用して XML ドキュメントを読み込む処理に似ています。

複雑さを優先

Java EE 仕様の初期バージョンは、開発者から非常に複雑で扱いづらいという批判を多く受けました。**「複雑」か「単純」かで言えば、**「複雑」の方向に進んでいたのは明らかでした。

WebSocket

Java EE 開発において、全二重双方向通信を要求する Web クライアントや Web クライアント・アプリケーションを扱う機会が増加しています。 このような要求の多くは、IETF RFC 6455 で定義された WebSocket プロトコルを活用しようとする HTML5 ベースの Web サイトによるものです。 また、JavaFX アプリケーションなどの Web ベースではないアプリケーションでも、さまざまな状況で WebSocket プロトコルのメリットを享受できます。

開発者は、JSR 356（Java API for WebSocket）を Java EE 7 プラットフォームの一機能として利用できます。

JSR 356 は Java EE 開発者に、WebSocket 管理のための使いやすいアノテーションベースのフレームワークを提供します。JSR 356 を使用しない場合、開発者は WebSocket エンドポイントの登録や別の WebSocket エンドポイントへの接続のために、定型的なコードを大量に記述する必要があります。

WebSocket プロトコルの内部的な詳細はすべて、JSR 356 の実装によって提供されます。そのため、開発にかかる時間を大幅に削減できます。WebSocket エンドポイントの登録や受信メッセージの取得は、**リスト 5** のようにして実行できます。

リスト 5 のコードから分かるように、WebSocket のライフサイクル管理を含めて、処理の大部分が Java EE プラットフォームの実装によって実行されます。@ServerEndpoint("/myEndpoint") アノテーションを付加した WebSocket にクライアントが接続すると、@OnOpen アノテーションを付加したメソッドが呼び出され、Session オブジェクトが渡されます。メッセージを受信するたびに、@OnMessage アノテーションを付加したメソッドが呼び出されます。

接続が閉じられたときには、@OnClose アノテーションを付加したメソッドが呼び出されます。

開発者にとって WebSocket API の利用は、アプリケーション固有のロジックを実装し、各メソッドに適切なアノテーションを付加するだけでほぼ終了する単純な作業です。 WebSocket の他の例についてはこちらを参照してください。

リスト3　リスト4　リスト5

```
JsonParserFactory factory = Json.createParserFactory();
 JsonParser parser = factory.createParser(...);
while (parser.hasNext()) {
Event event = parser.next();
}
```

すべてのリストのテキストをダウンロード

並行処理

Java EE 7 より前のバージョンでは、Java SE 向けに開発した並行処理用のツールを Java EE の実装で使用できないという不満がたびたび聞かれました。 Java EE では、すべてのリソースがコンテナによって管理されます。 一方、java.util.concurrent パッケージに含まれるスレッドやエグゼキュータは、コンテナによって管理されてはいません。 そのため、スレッドやエグゼキュータをエンタープライズ・アプリケーションで使用することは危険であり、禁止される場合もあります。

このような状況下で、一般的な Java 開発者の作業は困難を極めました。なぜなら、Java SE アプリケーションの開発で使用する並行処理用のツールに慣れているのに、同じ考え方を Java EE アプリケーションの開発に適用できないためです。 しかし、JSR 236 によって、コンテナによって管理される環境でも従来の並行処理の考え方を適用できるインフラストラクチャが整いました。

java.util.concurrent.ExecutorService をベースとする

ManagedExecutorService は、コンテナによって管理され、Java EE アプリケーションで安全に使用できる ExecutorService です。
ManagedExecutorService の実装は、JNDI を使用して取得し、簡単に挿入できます（**リスト 6**）。

Java EE 7 のその他のアップグレード

新規の JSR が導入された以外にも、Java EE 7 では多くの既存仕様が大幅にアップグレードされています。 特に重要な点は、Java Message Service（JMS）仕様が 1.1 から 2.0 に変更されたことです。 また、Java EE 6 で導入された JAX-RS 仕様もバージョン 2.0 に更新され、Web Profile の一機能になりました。

JMS：Java EE 5 と Java EE 6 の仕様では、Enterprise JavaBeans（EJB）や永続化リソースといったエンタープライズ・リソースの大部分が非常に扱いやすくなりました。 ただし、メッセージング・コンポーネントは、バージョン 1.1 がリリースされた 2003 年以降は変更されていませんでした。 そのため、Java EE 6 の他のテクノロジーと比較すると、JMS 1.1 の使用には定型的なコードや設定が大量に必要でした。

しかし、JMS 2.0 で、他のコンポーネントと同様の簡素化が実施されました。 メッセージの送信や処理に必要となるコードと設定がデフォルトで削減されている一方で、アノテーションやディスクリプタを使用した、より柔軟で複雑な設定も可能です。

JMS は基本的に、複数のソフトウェア・コンポーネント間のメッセージ送信を扱います。
メッセージ送信は単純な処理ではなく、さまざまなアクターが実行する多くの中間ステップが必要です。 JMS 1.1 では、これらの中間ステップの大部分を Java 開発者が実装する必要がありました。 しかし、JMS 2.0 では、中間ステップの大部分が JMS 実装により提供されます。

常に進化

エンタープライズ・コンピューティングの世界は常に進化しており、Java EE 標準も同様に進化しなければなりません。

JMS 1.1 の典型的な使用法は、ConnectionFactory と ConnectionQueue をインジェクションし、Connection、Session、MessageProducer、Message を作成するというものです。 JMS 2.0 ではデフォルトで、この使用法が大幅に簡素化されています。 JMSContext を作成し、このコンテキストを使用してメッセージを送信します（**リスト 7**）。

JAX-RS：Java API for RESTful Web Services は、Java EE の世界でもっとも急速に進化している API の 1 つです。 いうまでもなく、この背景には、REST ベースの Web サービスが広範に採用され、そのような Web サービスを利用するアプリケーションも増加しているという状況があります。

JAX-RS 2.0 では、REST API のサーバー側の仕様が見直され、さらにクライアント側の仕様が追加されています。 今日のエンタープライズ・アプリケーションは REST API を介してサービスを公開している場合が多い一方で、他の（外部）Web サービスを利用する必要もあります。 Java EE 6 で他の Web サービスを利用するためには、開発者自身で定型的なコードを作成するか、独自仕様のフレームワークを利用する必要がありました。

JAX-RS では、javax.ws.rs.client パッケージを定義しており、このパッケージを利用して外部 Web サービスに容易に問い合わせることができます。
たとえば my.server.com という架空のサーバーに対する POST リクエストは、**リスト 8** のように実行できます。

リスト 8 ではメソッド・チェーンを使用しています。メソッド・チェーンは、多数のメソッドを呼び出す必要がある（またはその可能性がある）場合に非常に便利で生産的な手法です。

JAX-RS 2.0 仕様では、クライアントのリクエストの結果を java.util.concurrent.Future として取得できます。そのため、非同期リクエストを実現することも可能です。 非同期リクエストは、Target インスタンスに対して async() メソッドの呼出しを追加するだけで実行できます。非同期処理はサーバー側でも実装できます。その場合は、処理メソッドに @Suspended Asyncresponse シグネチャを追加します。

リスト6　リスト7　リスト8

```
ManagedExecutorService mes = (ManagedExecutorService)ctx.lookup(
"java:comp/env/concurrent/ThreadPool");
```

すべてのリストのテキストをダウンロード

まとめ

開発においてビジネス・ニーズと開発生産性のバランスをとることは難題ですが、Java EE 仕様はこれを打破する必要があります。 Java EE 5 以降、開発生産性の優先度が上がり、ビジネス・ニーズを脅かすことなく生産性を向上させるためのさまざまな変更が実施されました。

Java EE 7 もこの方針に従って、今日のビジネス環境において非常に重要な意味をもつ新機能を追加するとともに、既存の仕様の一部を簡素化しています。これにより、Java EE 7 仕様では以下の 2 つの目標を達成しています。

- コードをほとんど記述しなくても、デフォルト動作を容易に利用できるようにすること
- デフォルト動作と異なる場合も、必要としている動作を設定できるようにすること

既存のアプリケーション・サーバーの大半が Java EE 7 仕様を実装する見込みです。 GlassFish プロジェクトがさまざまな仕様のリファレンス実装の役割を果たしています。 **</article>**

LEARN MORE

- Java EE 7仕様
- 『Batch Applications in Java EE 7—Understanding JSR 352 Concepts』
- 『JSON-P: Java API for JSON Processing』
- 『JMS 2.0 Early Draft—Simplified API Sample Code』
- JMS 2.0のブログ

EDUARDO MORANCHEL、EDGAR MARTINEZ

BIO

Eduardo Moranchel、Edgar Martinez（@edmtzx）：共に Oracle Mexico Development Center の Java カリキュラム開発者。Oracle University の Java 関連のコースやチュートリアルの作成に従事。Guadalajara JUG の共同主催者として、メキシコの Java ユーザー・グループ拡大を推進。

Java EE 7

HTML5の活用

次世代インターネット・アプリケーションを作成するための新機能を探究する

Java EE 7 では、WebSocket、HTML5、JSON など、発展し続ける最先端の標準テクノロジーが、それぞれ対応する API によりサポートされます。

本記事では最初に WebSocket の背景を述べた上で、これらの新しいプラットフォーム関連機能のデモを行う Web アプリケーションについて説明します。

注：本記事で紹介するサンプルのソース・コードは、Maven プロジェクトとしてこちらからダウンロードできます。

WebSocket の背景

近年の Web アプリケーションは双方向性の面で発展を遂げており、ブラウザと Web サーバーの間での通信の向上が求められています。 当初は、サーブレットとブラウザ内でのロングポーリングによって安定した通信を実現していましたが、この方法ではサーバー側で問題が見られました。Web コンテナのスレッド・モデルでは、応答を書き込むまでの間、すべてのリクエストを 1 つのスレッドで処理する必要があったためです。 その後、Java EE 6 に非同期サーブレットが導入され、リクエストを非同期的に処理できる、より効率的なアプローチが可能になりました。

ポーリングと Comet/Ajax を使用するアプローチは、サーバーとブラウザとの全二重通信を疑似的に実現するものです。 一方、WebSocket は、サーバーと Web ブラウザのどちらからでもデータ送信を開始できる全二重双方向通信を実現する API です。

WebSocket によって、クライアント・リクエストに対する Web サーバーの応答方法が変わります。 Web サーバーは接続を閉じずに、101 ステータスを送信して接続を開いたままにします。これにより、ストリームに書き込まれるメッセージを受け取ることも、Web サーバーからメッセージを書き込むことも可能になります。さらに、ストリームをやり取りするクライアントの側においても、書き込まれたメッセージを受け取ることも、メッセージを書き込むことも可能です。 そのため、Web サーバーとクライアントの両者が、ストリームに対する読取り / 書込みと通信をリアルタイムで実行できます。

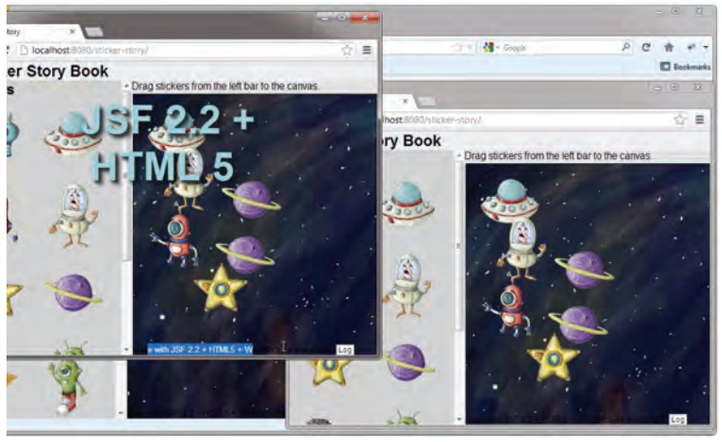
Java EE 7 より前のバージョンでは、WebSocket の実装方法は Web コンテナに依存していました。 たとえば、Tomcat 向けに WebSocket を実装した後に Jetty に移行するような場合は、Jetty 用に WebSocket を実装し直す必要がありました。

注：新しいテクノロジーのすべてに言えることですが、WebSocket の適用時期は Web ブラウザによって異なります。 ブラウザの WebSocket 対応の詳細については、こちらで確認できます。

ステッカー・ストーリー・アプリケーションの概要

それでは、単純なサンプルを使用して、前述のテクノロジーの動作を確認していきます。

本記事では、ステッカーをブック（キャンバス）にドラッグしてストーリーを作成する子供向け Web アプリケーション「ステッカー・ストーリー」について紹介します。このアプリケーションでは共同でストーリーを作成でき、ある子供のブックにステッカーが配置された瞬間に、そのステッカーが他の子供のブックにも描画されます。 このアプリケーション



このビデオでは、ステッカー・ストーリー・アプリケーションのデモを見ることができます。

図1

を実際に実行した様子については、ビデオをご覧ください。

図 1 は、ステッカー・ストーリー Web アプリケーションのホームページです。 ビデオで紹介されているとおり、左側のサイドバーからキャンバスにステッカーをドラッグ・アンド・ドロップすると、そのとき開いているすべての Web ブラウザでそのステッカーが描画されます。

ステッカー・ストーリー・アプリケーションの取得と実行

1. GlassFish v4 Promoted Build 79（glassfish-4.0-b79.zip）をダウンロードします。
   GlassFish の zip アーカイブはこちらからダウンロードできます。
2. ダウンロードした GlassFish を任意のディレクトリに解凍します。
3. NetBeans をダウンロードしてインストールします。 本記事で使用したバージョンは NetBeans 7.3 です。
4. ステッカー・ストーリーの Maven プロジェクト（sticker-story.zip）をダウンロードします。
5. ダウンロードしたプロジェクトを任意のディレクトリに解凍します。
6. NetBeans を起動し、**Services** タブで「**Servers**」ノードを右クリックし、次に「**Add Server**」をクリックします。
7. Server リストで「**GlassFish Server 3+**」を選択し、「**Next**」をクリックします。
8. **Server Location** フィールドに、GlassFish v4 を解凍したディレクトリのパスを入力し、「**Finish**」をクリックします。
9. 「**File**」→「**Open Project**」を選択します。
10. Open Project ダイアログ・ボックスで、ステッカー・ストーリーの Maven プロジェクトを開きます。
11. プロジェクトを右クリックして、「**Run**」を選択します。 プロンプトが表示されたら、先ほどインストールした GlassFish サーバー（GlassFish Server 3+）を選択します。

この時点で、Web ブラウザのアドレス欄に http://localhost:8080/sticker-story と入力することで、**図 1** のようなページが表示されます。 お使いのブラウザが WebSocket に対応していることをこのページで確認してください。

アプリケーションのコンテンツ

このアプリケーションには、以下のファイルと Java クラスが含まれます。

- index.html：ホームページ。JavaServer Faces（JSF）2.2 と HTML5 のコードを利用
- org.sticker.jsf.StickerSheet クラス：利用可能なすべてのステッカーをホームページに表示するためのマネージド Bean
- story-page.js：WebSocket クライアントおよびドラッグ・アンド・ドロップ機能が実装された JavaScript コード
- org.sticker.websocket.Sticker クラス：WebSocket によって送受信されるオブジェクト
- org.sticker.websocket.StickerEncoder クラス、org.sticker.websocket.StickerDecoder クラス：WebSocket によって送信されたデータをオブジェクトに変換するクラス
- org.sticker.websocket.StoryWebSocket クラス：WebSocket ハンドラ。 現在のストーリー・ブックへのステッカーの保存も実行

HTML5 と JSF 2.2 の組み合わせ -HTML との親和性の高いマークアップ

JSF 2.2（JSR 344）の仕様リードである Ed Burns がブログで説明しているように、JSF 2.2 の Friendly Markup（HTML5 との親和性の高いマークアップ）機能を使用することで、ページの作成者が HTML の描画方法を完全に制御できます。また、Friendly Markup 機能は、マークアップとビジネス・ロジックの分離も促進します。

例を見てみましょう。 ステッカー・ストーリー・アプリケーションの画面は、HTML5 と JSF を使用して作成します。 **リスト 1** に index.xhtml ファイルの内容を示します。index.xhtml ファイルの大部分は HTML5 で記述されていますが、ブラウザのサイドバーにステッカー画像を描画するため、JSF の <h:graphicImage> タグを使用しています。

- JSF 2.2 では HTML 属性をパススルーできます。 パススルーの方法は 2 つあります。 1 つは、パススルー属性用の名前空間（http://java.sun.com/jsf/passthrough）を使用すること、もう 1 つは、子の TagHandler f:passThroughAttribute（属性が複数ある場合は f:passThroughAttributes）を使用することです。 本記事のサン

プルでは、パススルー属性用の名前空間を利用しています。

リスト 1 の <h:graphicImage> タグに注目してください。 このサンプルの graphicImage タグには 3 つの属性（draggable、ondragstart、data-sticker）があります。これらの属性は graphicImage の属性ではありませんが、ステッカーを正常に動作させるために必要となります。

そのため、接頭辞 p: を付け、これら 3 つの属性が JSF のパススルー機能によって追加されるようにしています。

次の点に注意してください。 graphicImage は JSF のマークアップであるため、名前空間 http://java.sun.com/jsf/passthrough を使用しました。 しかし、JSF で認識されないマークアップ内に要素をパススルーする場合は、http://java.sun.com/jsf という名前空間を使用します。 この名前空間の例については、**リスト 1** の head タグや script タグで確認してください。

ご存じでしたか？
WebSocket は、サーバーだけではなく Web ブラウザも関与する、**全二重 / 双方向通信用の API です。**

リスト 2 は、ステッカーの名前を取得するためのマネージド Bean です。

@Named アノテーションによって、このマネージド Bean を JSF に公開しています。その後、index.xhtml（**リスト 1**）ファイル内の JSF ui:repeat タグで、値（value）として #{stickerSheet.allStickers} 式を指定し、この Bean のメソッドを呼び出しています。

ドラッグ・アンド・ドロップのコードの記述

HTML5 ではドラッグ・アンド・ドロップ機能が簡素化されており、特定のイベントを JavaScript コード内で処理する必要があります。このイベントには、ドラッグ対象オブジェクトに関するイベントと、受取り側オブジェクトに関するイベントの 2 種類があります。

ドラッグ対象オブジェクト（この例ではステッカー）では、ondragstart イベント（**リスト 1**）を定義し、JavaScript を使用して処理する必要があります。

前述のとおり、JSF 2.2 では、パススルー・パラメータを利用して ondragstart イベントを追加できます。 **リスト 3** の JavaScript コードは、ドラッグの開始時に実行する処理を示しています。

場合によっては、ドラッグ・アンド・ドロップ操作中に、コンポーネント間で転送されるデータを保管するための仲介オブジェクトを使用する必要があります。 そのために、この例の JavaScript では drag メソッドの event.dataTransfer プロパティにデータを格納しています。 このプロパティには文字列しか格納できないため、もっとも簡単なデータ格納方法は JSON を使用することです。

ドラッグ・イベント用の JavaScript コードでは、ステッカー表現を作成し、画面からステッカーのドラッグ先の座標を取得し、タグの data-sticker 属性からステッ

リスト1　リスト2　／リスト3

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:jsf="http://java.sun.com/jsf"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p=" http://java.sun.com/jsf/passthrough">
 <head jsf:id="head">
  <title>Sticker Story</title>
  <script jsf:target="body" jsf:name="story-page.js"/>
  <link jsf:name="styles.css" rel="stylesheet" type="text/css" />
 </head>
 <body jsf:id="body">
  <header>
   <h1>Sticker Story Book</h1>
</header>
  <nav>
   Drag stickers from the left bar to the canvas.
  </nav>
  <aside>
   <h2>Stickers</h2>
   <div id="stickerContainer">
    <ui:repeat var="imgName"
        value="#{stickerSheet.allStickers}">
     <h:graphicImage library="stickers" name="#{imgName}"
            style="float:left" p:draggable="true"
            p:ondragstart="drag(event)" p:data-
            sticker="#{imgName}" />
    </ui:repeat>
   </div>
  </aside>
  …
 </body>
</html>
```

すべてのリストのテキストをダウンロード

カー画像のファイル名を取得します。このファイル名の取得でも、HTML5 の別の機能を使用しています。HTML5 では、属性に接頭辞 data- を付加することで、アプリケーション固有のデータを格納するカスタム属性を使用できます。一方、JSF 2.2 では、そのようなカスタム属性を追加するために、パススルー属性を使用します。

ドロップ操作については、ドロップ先オブジェクト（つまり受取り側）で 2 つのイベント・ハンドラを定義する必要があります。

1 つは、ondragover イベント・ハンドラです。このイベント・ハンドラは、マウスのドラッグ・オーバー（ホバー操作）を処理して、ドラッグが有効かどうかを判定します。ドロップに対して詳細な検証を行うために、サンプル・コードでは ev.preventDefault() を呼び出しています。デフォルトの動作ではドロップが拒否されるためです。

もう 1 つは、ondrop イベント・ハンドラです。このイベント・ハンドラは、何らかの要素をドラッグ中にマウス・ボタンが離されたときのドロップ・イベントを処理します。このイベントを処理する JavaScript コード（**リスト 4**）では、デフォルトの動作（ドロップの拒否）を防止した後に、dataTransfer からデータを取得しています。転送される情報はテキストであるため、JavaScript の標準メソッドである JSON.parse メソッドを使用して、テキストをオブジェクトに変換する必要があります。次に、ドラッグされたオブジェクトを、サーバーに送信する実際のオブジェクトに変換します。**リスト 4** の drop メソッドでは、キャンバス上にドロップされたステッカーの左上隅の座標も計算し、さらにステッカーの名前をオブジェクトに追加しています。次に、このオブジェクトを JSON 文字列に変換し、最後に WebSocket.send メソッドを使用してこのオブジェクトを送信しています。

JavaScript で記述する WebSocket クライアント

ステッカー・ストーリー・アプリケーションは、ドラッグ・アンド・ドロップ操作が行われたときに、WebSocket を使用してデータをサーバーに送信します。それでは、WebSocket 接続はどのように確立するのでしょうか。例を見てみましょう。

ページの読込み後、initialize メソッド（**リスト 3**）が呼び出されます。このメソッドでは最初に、HTML5 キャンバスを使用してアプリケーションの背景を描画します。

次に、socket = new WebSocket("ws://localhost:8080/sticker-story/story/notifications") という文を使用して WebSocket 接続を開き、通信の確立と開始を行います。HTTP の代わりに WebSocket プロトコルを使用している点と、絶対パスを使用している点に注意してください。

注：Web サーバーを別のポートで実行している場合は、この行をお使いのサーバーのポート番号で変更する必要があります。

リスト 3 のその次の行では、サーバーがブラウザにメッセージをプッシュしたときにクライアントで呼び出されるメ

リスト4

```
function drop(ev) {
 ev.preventDefault();
 var bounds = document.getElementById("board")
.getBoundingClientRect();
 var draggedText = ev.dataTransfer.getData("text");
 var draggedSticker = JSON.parse(draggedText);
 var stickerToSend = {
  action: "add",
  x: ev.clientX - draggedSticker.offsetX - bounds.left,
  y: ev.clientY - draggedSticker.offsetY - bounds.top,
  sticker: draggedSticker.sticker
 };
 socket.send(JSON.stringify(stickerToSend));
 log("Sending Object " + JSON.stringify(stickerToSend));
}

// Web socket on message received:
function onSocketMessage(event) {
 if (event.data) {
  var receivedSticker = JSON.parse(event.data);
  log("Received Object: " + JSON.stringify(receivedSticker));
  if (receivedSticker.action === "add") {
   var imageObj = new Image();
   imageObj.onload = function() {
    var canvas = document.getElementById("board");
    var context = canvas.getContext("2d");
    context.drawImage(imageObj,
receivedSticker.x, receivedSticker.y);
   };
   imageObj.src =
"resources/stickers/" + receivedSticker.sticker;
  }
 }
```

すべてのリストのテキストをダウンロード

ソッドを定義しています。

次に、**リスト 4** の onSocketMessage メソッドについて見てみましょう。

onSocketMessage メソッドは、サーバー側からの WebSocket メッセージを処理するためのメソッドです。

ステッカー・ストーリー・アプリケーションは、WebSocket 接続により、JSON オブジェクトをテキストとして送信します。 そのため、メッセージの受信時にテキストを Sticker オブジェクトに変換し、onSocketMessage メソッドで以下の処理を実行します。

- データの変換
- ステッカーの画像の取得
- ステッカーの座標を使用し、キャンバス上に画像を描画

さらに、この時点で接続は開いたままであるため、サーバーへのメッセージ送信は、WebSocket JavaScript オブジェクトの send メソッドを使用して実行します。 本当に簡単です。

リアルタイム通信のための WebSocket

Java EE 7 での WebSocket の作成は非常に簡単です。 いくつかのアノテーションを作成するだけで、WebSocket の設定が完了します。

ステッカー・ストーリー・アプリケーションにおいて、WebSocket として設定されたクラスは StoryWebSocket クラスです（**リスト 5a**、**5b**）。

クラス宣言の最初に @ServerEndpoint アノテーションを付加しています。 サーブレットの場合と同様に、マッピングを定義する必要がありますが、WebSocket では、@ServerEndpoint の値により、その WebSocket のマッピングを定義できます。 そのため、以下のコードを使用するだけで WebSocket を宣言できます。この場合、サーバーは /websocketMapping URL において、この WebSocket への接続を待機します。

```
@ServerEndpoint(
"/websocketMapping")
```

現時点では、encoders 属性と decoders 属性については考えず、先にクラスのメソッドについて説明します。WebSocket の各イベントを処理するメソッドを定義する際にもアノテーションを使用します。

@OnOpen、@OnClose、@OnError は、WebSocket 接続のライフサイクルに関連するアノテーションです。 @OnOpen を付加したメソッドは接続を開いたときに、@OnClose を付加したメソッドは接続を閉じたときに、@OnError を付加したメソッドは接続でエラーが発生したとき（例外が処理されなかったとき）に呼び出されます。

サンプル・アプリケーションでは、@OnOpen アノテーションを使用して、接続中のユーザーにステッカーをプッシュしています。 この方法によって、新しく接続したユーザーは、他のユーザーがそれまでにキャンバス上に配置したすべてのステッカーを受け取ります。

@OnMessage アノテーションは WebSocket 実装の核心です。 @OnMessage アノテーションが付加されたメソッドは、クライアントがメッセージを送信するたびに呼び出されます。サ

リスト5a　リスト5b

```
package org.sticker.websocket;

import java.io.IOException;
import java.util.*;
import java.util.logging.*;
import javax.websocket.EncodeException;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;

@ServerEndpoint(
      value = "/story/notifications",
      encoders = {StickerEncoder.class},
      decoders = {StickerDecoder.class})
public class StoryWebSocket {

 private static final List<Sticker> stickers =
Collections.synchronizedList(new LinkedList<Sticker>());

@OnOpen
 public void onOpen(Session session) {
  synchronized (stickers) {
   for (Sticker sticker : stickers) {
    try {
     session.getBasicRemote().sendObject(sticker);
    } catch (IOException | EncodeException ex) {
     Logger.getLogger(StoryWebSocket.class.getName()).log
(Level.SEVERE, null, ex);
    }
   }
  }
 }
```

すべてのリストのテキストをダウンロード

ンプル・アプリケーションでは、クライアントがステッカーを送信すると、追加されたステッカーが WebSocket によって保存され、接続中のすべてのユーザーにプッシュされます。

@onMessage メソッドでは Sticker オブジェクト（**リスト 6**）を引数として受け取り、さらに以下のメソッドを使用して Sticker オブジェクトを他のユーザーに書き込んでいます。

session.getBasicRemote() .sendObject()

前述のとおり、WebSocket で送受信される情報はテキストです。 ではなぜ、サンプルのメソッドは Sticker オブジェクトを受け取り、セッションに Sticker オブジェクトを書き込んでいるのでしょうか。その答えは、@ServerEndpoint 宣言の中にあります。具体的には、encoders 属性と decoders 属性です。

WebSocket では、デコーダを使用してテキスト・メッセージをオブジェクトに変換し、そのオブジェクトを対応する @OnMessage メソッドで処理できます。また、オブジェクトがセッションに書き込まれたときには常に、エンコーダを使用してオブジェクトをテキストに変換し、そのテキストをクライアントに送信できます。

URL パスのパラメータ： @ServerEndpoint マッピングに URL パスのパラメータを追加することで、URL パスのパラメータを設定できます。 そのためには、以下のように、パラメータ名を波括弧で囲みます。

/pathToWebSocket/ {parameter1}

URL パスのパラメータを使用するためには、アノテーションを付加したメソッドに文字列パラメータを追加し、このメソッド・パラメータに @PathParam("parameterName") アノテーションを追加します。

たとえば、**リスト 5a** と **5b** の例に、@PathParam("parameter1") という文字列パラメータを追加することが可能です。

パラメータは必要な数だけ追加できます。

また、WebSocket 関連のアノテーションが付加されたメソッドは、WebSocket セッションをパラメータとして受け取ることができます。 このサンプル・アプリケーションでは WebSocket セッションをパラメータとして渡していますが、このパラメータは必須ではないため省略することもできます。

@OnMessage アノテーションを付加したメソッドには、他にパラメータを 1 つ指定できます。 このパラメータは受信中のメッセージを表すもので、@ServerEndpoint アノテーションで宣言されたデ

本当に簡単

Java EE 7 での WebSocket の作成は非常に簡単です。いくつかのアノテーションを作成するだけで、WebSocket の設定が完了します。

リスト6

```
package org.sticker.websocket;

public class Sticker {
 private int x;
 private int y;
 private String image;

 public Sticker() {
 }
 public int getX() {
  return x;
 }

 public void setX(int x) {
  this.x = x;
 }

 public int getY() {
  return y;
 }

 public void setY(int y) {
  this.y = y;
 }
 public String getImage() {
  return image;
 }
 public void setImage(String image) {
  this.image = image;
 }
}
```

すべてのリストのテキストをダウンロード

コーダを使用してデコードされた任意のオブジェクト形式か、デコーダが指定されていない場合は String 形式になります。

新しい JSON API と WebSocket との併用

このサンプルでは、Java EE 7 に新しく搭載された Java API for JSON Processing（JSR 353）を使用しています。

ステッカー・ストーリー・アプリケーションは、オブジェクトの JSON 文字列表現を使用して、クライアントおよび WebSocket サーバーと通信します。WebSocket ではデコーダとエンコーダを使用して、WebSocket に送信されたテキストを Sticker オブジェクトに透過的に変換し、また、クライアントに送信する Sticker オブジェクトをテキストに変換します。

StickerDecoder クラスと StickerEncoder クラス（**リスト 7**、**リスト 8**）では、以下のように JSON API を使用しています。

- ソケット・ストリームを読み取ってオブジェクトを取得
- オブジェクトを JSON 文字列表現に変換してソケットに書き込み

WebSocket からデータを読み取る際の形式に応じた適切なデコーダを作成できます。 同様に、データをソケットに書き込む方法に応じたエンコーダを作成できます。

リスト 7 に示すように、デコーダでは親の Decoder インタフェースに定義されているすべてのインタフェースを実装する必要があります。 さまざまな種類のデコーダやエンコーダがあり、この種類に応じて decode メソッドと encode メソッドのパラメータが変わります。

リスト 7 の StickerDecoder クラスでは、Decoder.TextStream インタフェースを使用しています。このインタフェースにより、Reader を使用してソケットからデータを読み取ることができます。

文字列を JsonObject に変換するためには JsonReader を使用します。JsonReader は、Reader を渡すことで簡単に作成できます。

注： JsonReader オブジェクトは直接作成せず、代わりに JsonProvider クラスを使用してリーダーおよびライターを作成してください。

リスト 8 に、オブジェクトを WebSocket ストリームに書き込むための StickerEncoder クラスを示します。

まとめ

WebSocket テクノロジー、JSF の HTML5 パススルー・パラメータ、JSON オブジェクトのシリアライズ処理の 3 つは、Java EE 7 の強力な追加機能であり、ブラウザと Web サーバーとの間で安定した通信が必要なアプリケーションの開発を大幅に簡素化します。

本記事では、Web アプリケーションを構築して、HTML との親和性の高い JSF 2.2 マークアップと HTML5 を組み合わせて使用する方法を示しました。

また、アプリケーションの WebSocket クライアントとドラッグ・アンド・ドロップ機能を JavaScript で実装しました。 最後に、新たに導入された Java API for WebSocket と Java API for JSON Processing を使用して、WebSocket サーバーを構築しました。 本記事で示したサンプルは、これらの新しいテクノロジーが持つ潜在能力のほんの一部分です。

謝辞： ステッカー・ストーリー・アプリケーションのイラストを描いてくださったメキシコの若手アーティスト、Marco Velasco 氏に心から感謝いたします。 **</article>**

リスト7　リスト8

```
package org.sticker.websocket;

import java.io.IOException;
import java.io.Reader;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.json.spi.JsonProvider;
import javax.websocket.DecodeException;
import javax.websocket.Decoder;

public class StickerDecoder implements
Decoder.TextStream<Sticker> {

 @Override
 public Sticker decode(Reader reader) throws DecodeException,
IOException {
  JsonProvider provider = JsonProvider.provider();
  JsonReader jsonReader = provider.createReader(reader);
  JsonObject jsonSticker = jsonReader.readObject();
  Sticker sticker = new Sticker();
  sticker.setX(jsonSticker.getInt("x"));
  sticker.setY(jsonSticker.getInt("y"));
  sticker.setImage(jsonSticker.getString("sticker"));
  return sticker;

 }
}
```

すべてのリストのテキストをダウンロード

LEARN MORE

- JSF 2.2 JSR
- Java API for WebSocket JSR
- Java API for JSON Processing JSR
- Java EE 7とWebSocket APIに関するArun Guptaのビデオ

ADAM BIEN

BIO

JavaFX Scene Builderを使用する効果について、著者**Adam Bien**の説明を聞く

Adam Bien：コンサルタント兼ライターであり、Java EE 6 および 7、EJB 3.X、JAX-RS、CDI、JPA 2.X の各 JSR の Expert Group メンバー。 Java のアーキテクト兼開発者。Java Champion、JavaOne 2009 および 2011 Rock Star。『Real World Java EE Patterns—Rethinking Best Practices』および『Real World Java EE Night Hacks—Dissecting the Business Tier』の著者。

写真：
THOMAS EINBERGER/
GETTY IMAGES

パート1

JavaFX Scene Builderのエンタープライズ・アプリケーションへの統合

JavaFX Scene Builderで作成した結果を複数のビューから成るエンタープライズ・アプリケーションに統合するための実用的な方法

WYSIWYG（What-You-See-Is-What-You-Get、「見たままが手に入る」）方式の Java エディタは、90 年代半ばに登場して以来、迅速な開発や生産性の向上を約束してきました。 通常、UI の構築は、コードの自動生成と変更不可能な領域を基盤としていました。 開発者は自動生成されたコードが正しいと信じるしかなく、モデリングに誤りがあれば UI が破損することもよくあり、落胆したものです。 自動生成されたコードを分離する明確な方針がなければ、WYSIWYG による開発も、すぐに保守不可能で、各種ベンダーの独自仕様の Java エディタに大きく依存したものとなってしまいます。

JavaFX 2 には Scene Builder が付属しています。Scene Builder は、UI を視覚的に作成するためのツールです。 本記事では、Scene Builder で作成した結果を、複数のビューから成るエンタープライズ・アプリケーションに統合する実用的な方法について、作成 / 読取り / 更新 / 削除（CRUD）のサンプルを使用しながら説明します。

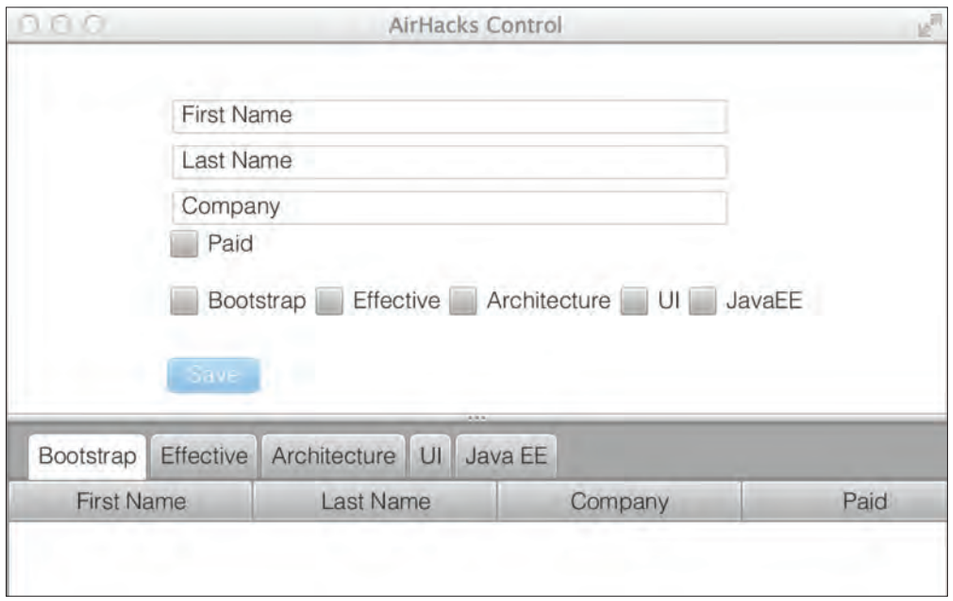


図1

サンプルの CRUD アプリケーション「AirHacks Control」（**図 1**）は、数日間に及ぶイベントの参加者情報を簡単に管理できるアプリケーションです。 参加者は登録した日程に応じて特定のタブに表示されます。 すでに登録されている参加者の情報は表や入力領域を使用して管理できます。 データはローカルの Apache Derby データベースに保存されますが、このデータベースは、全体的な設計はそのままでサーバーにも簡単に移行できます。

注：AirHacks Control アプリケーションのソース・コードは、こちらからダウンロードできます。

エンタープライズの意味

コンシューマ・アプリケーション（一般消費者向けアプリケーション）とエンタープライズ・アプリケーション（企業向けアプリケーション）の境界は曖昧になってきていますが、エンタープライズ・アプリケーションでは依然として、多様な要件とプロセスを扱います。 一方、成功を収めるコンシューマ・アプリケーションは、1 つの問題に特化し、その問題を可能な限り洗練された方法で解決するという傾向にあります。 ブランティングや、1 ピクセルもずれることのない位置設定、さらには美しいドック・アイコンさえも、コンシューマ・アプリケーションの商業的な成功には大きく寄与します。

エンタープライズ・アプリケーションも不格好である必要はないのですが、洗練された外観が第 1 目標ではありません。 もっとも重要とされるのは、長期的な保守性が確保され、ユースケースやユーザー・ストーリーがビュー（画面）にそのまま反映されることです。

また、エンタープライズ・アプリケーションでは、REST、SOAP、IIOP や、その他の独自仕様のプロ

トコル経由で公開されるサーバー上に、すべてのビジネス・ロジックを配置します。一方、コンシューマ・アプリケーションでは、各種クラウド・プロバイダが提供することも多いビジネス・ロジックをより多く含めて、バックエンドとの通信頻度を減らす傾向にあります。

成功を収めるコンシューマ・アプリケーションの UI は、デザイナーやユーザビリティの専門家の影響を大きく受けますが、エンタープライズ・アプリケーションの UI は通常、ドメインの専門家やユーザーからの機能拡張要求（RFE）を直接反映したものになります。

本記事では、一般的に大規模な開発チームを要し、その結果として、より簡潔で明快な構成が求められる、エンタープライズ・アプリケーションに焦点を絞ります。

シーンを作る

JavaFX 2 では Scene Builder が利用できます。 Scene Builder は、ユーザー・インタフェースを視覚的に作成するツールです。

Scene Builder の XML の本質

DRY（Don't Repeat Yourself、「重複の排除」）原則によって、XML は実用的な Java EE アプリケーションから姿を消しました。 影響を受けるクラスに直接アノテーションを配置することで、重複なしで必要なメタデータを管理できるようになり、Java EE は極めて生産的になりました。Java EE での開発を始めるためには、いくつかのアノテーションのみを知っていれば済みます。

Scene Builder の大部分は XML に基づいています。 UI 定義は、FXML と呼ばれる XML 方言で管理されます。 ただし、XML を使用しているものの、重複は発生しません。 実際、生成される FXML は「唯一の真実（Single Truth）」であり、DRY 原則に従っています。 また、平均的な Java EE アプリケーションと一般的な UI との最大の相違点は、必要となる設定やメタデータの量にあります。

一般的な UI では平均的な Java EE アプリケーションよりも利用できる規約の数が少なく、1 つの単純なビューでも、色、レイアウト、位置、サイズの設定が必要になります。そのため、一般的な UI の設定量は、あらゆる Java EE アプリケーションの設定量を大きく上回ります。

FXML ファイルは、実際には javafx.fxml.FXMLLoader ファクトリの設定情報に過ぎません。 FXMLLoader は FXML ファイルを読み取り、ルート・ノード（javafx.scene.Parent）を作成します。さらに、依存性注入（DI）フレームワークの興味深い拡張機能も備えていますが、その点については後ほど説明します。

Parent は手動ではなく、FXMLLoader を使用して自動で作成できます。FXMLLoader の全体的な設定は、Scene Builder で FXML ファイルを使用して実行されます。 JavaFX のノードは階層的に構造化されます。**リスト 1** から分かるように、FXML にはビュー構造がそのまま、しかも理解しやすい形式で反映されます。

FXML は理解しやすいだけでなく、情報を網羅しています。 FXML ファイルから Java コードを生成することも可能なほどです。

リスト1

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<AnchorPane id="AnchorPane" prefHeight="243.0" prefWidth="600.0"
styleClass="mainFxmlClass" xmlns:fx="http://javafx.com/fxml"
fx:controller="...">
 <children>
  <VBox id="VBox" fx:id="inputBox" alignment="CENTER"
layoutX="100.0" layoutY="33.0" prefWidth="343.0" spacing="5.0">
   <children>
    <TextField fx:id="firstName" prefWidth="200.0"
promptText="First Name" />
   <CheckBox fx:id="paid" mnemonicParsing="false"
prefWidth="103.0" text="Paid" />
   </children>
  </VBox>
  <Button fx:id="saveButton" defaultButton="true"
disable="true" layoutX="98.0" layoutY="192.0" onAction="#save"
prefHeight="22" prefWidth="57.0" text="Save" />
  <HBox id="HBox" alignment="CENTER" layoutX="100.0"
layoutY="149.0" spacing="5.0">
   <children>
   <!-- .... -->
   </children>
  </HBox>
 </children>
 <stylesheets>
  <URL value="@attendeeinput.css" />
 </stylesheets>
</AnchorPane>
```

すべてのリストのテキストをダウンロード

制御の課題、制御の反転

Scene Builder を使用すれば、ノード・ツリーを FXML から完全に自動生成してインスタンス化できます。 通常、FXML にビジネス・ロジックは含まれません。スクリプトによるイベント・ハンドラは必須ではなく、「主流」の方法でもありません。 それでは、Java で実装するロジックとFXMLとを関連付けるものは何でしょうか。

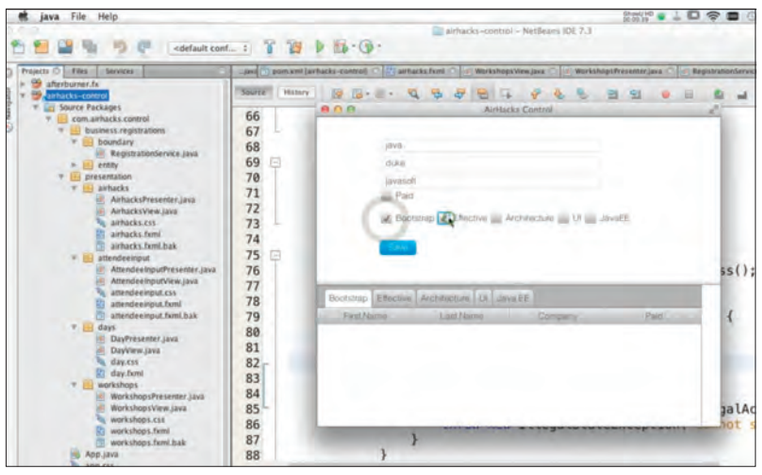
FXML は、任意の（デフォルト・コンストラクタを持つ）POJO を FXMLLoader で読み込んで作成し、宣言的に UI イベントにバインドできるという、単純ですが優れたアイデアを採用しています（**リスト 2**）。 読み込むクラスのメソッドは、Scene Builder 内の UI イベントに関連付けることができます（**リスト 3**）。

メソッド名は、FXML ファイル内の属性の値と対応付ける必要があります。 また、必須ではありませんが、@FXML アノテーションを付加することで、あらゆる UI コンポーネントを POJO にインジェクションすることもできます（**リスト 4**）。

この場合も、インジェクションされるコンポーネントの名前は、FXML ファイルの fx:id 属性の値と対応付ける必要があります。

従来の WYSIWYG ビルダーはビュー用のコード一式を生成するものであり、生成されたコードをその他のプレゼンテーション・ロジックと統合する作業が必要でした。 一方、Scene Builder は制御の反転のアプローチに従い、生成された XML ファイルを基に、プレゼンテーション・ロジックをインスタンス化します。また、手動で作成したコードへのバインディングも自動生成されます。そのため、開発者が Scene Builder の存在を意識することはほぼありません。

Scene Builder では、手動で作成したコードは生成されたアーティファクトに依存しません。依存関係はむしろその逆で、手動で作成したコードへのバインディングが FXML に含まれます。 この関係は、命名規則に基づいて設定されます。



Adam BienがAirHacks Controlアプリケーションの概要を説明します。

リスト2　リスト3　／リスト4

```
<AnchorPane (...) fx:controller="com.(...).AttendeeInputPresenter">
```

すべてのリストのテキストをダウンロード

設定より規約

コンシューマ・アプリケーションとエンタープライズ・アプリケーションの相違点としてはほかに、開発プロセスに関係する開発者や役割の数も挙げられます。コンシューマ・アプリケーションは、時には驚くほど少数の人員で開発されます。一方、平均的なエンタープライズ・アプリケーションでは、コンシューマ・アプリケーションより数桁多い人員が関わることもあります。

開発者が多くなるほど、構造や規約も増やす必要があります。 名称から判断できるような分かりやすいルールを策定することが、生産性と保守性の向上につながります。 個別のビューやフォームの作成方法を再現できるレシピがあれば理想的です。

ここで、サンプルの AirHacks Control アプリケーションに、1 つの規約を取り入れます。 この規約は、AttendeeInput という名前のビューであれば同名のパッケージ（attendeeinput）に属し、さらに以下のアーティファクトを含むというものです。

- AttendeeInputPresenter：手動で作成するプレゼンタクラス。FXMLLoader によって読み込まれる。すべてのプレゼンテーション・ロジックを含み、自動生成されるアーティファクトには依存しない
- AttendeeInputView：FXMLLoader をラップして使いやすくしたクラス。FXMLView を継承するが、自動生成されるアーティファクトには依存しない
- attendeeinput.css：ルック・アンド・フィールを管理するためのスタイルシート（初期状態では空）
- attendeeinput.fxml：Scene Builder によって自動生成される設定ファイル。手動で作成した AttendeeInputPresenter へのバインディングを含む

FXMLLoader を使用して FXML ディスクリプタから javafx.scene.Parent のサブクラスを作成する方法は、簡単ですが繰り返しが生じます。

そこで、FXMLView クラスでビューの作成をカプセル化し、さらに前述の規約を実装します（**リスト 5a**、**5b**）。

FXMLView は、Parent およびプレゼンタ POJO にアクセスするためのメソッドを提供し、さらに必要なすべてのファ

イルの読込みも行います。

規約を適用できる場合、ビューのラッパー・クラスは空のままにしておくことができます（**リスト 6**）。ただし、空であっても、このサブクラス自体は必要です。前述のすべての規約はクラス名に基づいており、CSS ファイルと FXML ファイルも同じパッケージから読み込まれます。

これで、わずか 3 行のコードでビューをインスタンス化し、プレゼンタを取得できます（**リスト 7**）。AttendeeInputView の利用箇所は、FXML、CSS、そして Scene Builder から完全に分離されます。

Guice でもう一歩踏み込む

現時点ではまだ、ビューにロジックを実装していません。そして、FXML によりインスタンス化される POJO にはプレゼンテーション・ロジックのみを含めるべきです。この POJO（つまりプレゼンタ）からアクセスする必要のあるエンタープライズ・サービスがいかなる種類のものであっても、このサービスのラッパー（あるいは、実際のビジネス・ロジックの実装でも）は、インスタンス化し、初期化してからプレゼンタに渡す必要があります。

Singleton は、サービス・ラッパーとしては疑う余地のない選択肢ですが、欠点もあります。Singleton は一度インスタンス化すると「未初期化状態に戻す」ことが難しく、そのためテストが困難です。通常、単体テストは、明確に定義された初期状態に基づいて実行されます。

本記事の CRUD サンプルでは、JavaFX アプリケーションが Java Persistence API（JPA）2 の EntityManager に直接アクセスします。この EntityManager は、インスタンス化した後に、次回以降の呼出しのためにキャッシュする必要があります。そこで、CRUD のすべての機能を、RegistrationService という 1 つのクラス内に実装します（**リスト 8** 参照。リスト 8 では EntityManager ラッパーを使用してユーザビリティを向上しています）。

RegistrationService は、Singleton ではなく単純な POJO として実装しています。単体テストの際には、RegistrationService#init メソッドの呼出しを省略し、テストの目的で EntityManager と EntityManagerFactory の両方をモックに置き換えることができます。ただし、RegistrationService 全体をモックに置き換え、モックのバックエンドを使用して UI をテストするとさらに良いでしょう。また、モック可能なビジネス・ロジックを使用することで、期待される動作を事前に登録し、バックエンドとの通信全体を省略することも可能です。

幸いなことに、JavaFX 2 では DI フレームワークのコールバック機能が導入されています。サンプルでは、FXMLLoader が DI フレームワークにクラスを渡すと、そのインスタンスが返されてインジェクションが実行されます。ビューやプレゼンタの作成は、FXMLLoader で実行し、抽象クラスの FXMLView 内でカプセル化しています。また、FXMLView は、FXMLLoader を計測する場所としても自然です。**リスト 9** では、ControllerFactory によって DI を実現しています。

InjectorProvider ユーティリティ（**リスト 10**）は、Guice Injector の簡易ラッパーです。

リスト5a　リスト5b　リスト6　リスト7　リスト8

```
public abstract class FXMLView {
    public static final String DEFAULT_ENDING = "view";
    protected FXMLLoader loader;

    public FXMLView() {
        this.init(getClass(), getFXMLName());
    }

    private void init(Class clazz, String conventionalName) {
        final URL resource = clazz.getResource(conventionalName);
        this.loader = new FXMLLoader(resource);
        this.loader.setControllerFactory(new Callback<Class<?>,
Object>() {
            @Override
            public Object call(Class<?> p) {
                return InjectorProvider.get().getInstance(p);
            }
        });
        try {
            loader.load();
        } catch (Exception ex) {
            throw new IllegalStateException(
"Cannot load " + conventionalName, ex);
        }
    }
    public Parent getView() {
        Parent parent = this.loader.getRoot();
        addCSSIfAvailable(parent);
        return parent;
    }
```

すべてのリストのテキストをダウンロード

Guice の DI インジェクションは、com.google.inject.Module インタフェースの実装クラスによって、型保証として実装されます。 サンプルでは、PersistenceModule クラス（**リスト 11**）が RegistrationService のインスタンス化とライフサイクル管理を担当します。

PersistenceModule は、RegistrationService のインスタンス化と初期化を行います。 作成されたインスタンスに RegistrationService.class がバインドされるため、結果的に RegistrationService は Singleton となりますが、テストのしやすさは損なわれません。

これで、標準的な @javax.inject.Inject を使用して、RegistrationService を任意のプレゼンタにインジェクションできます。インジェクションされた RegistrationService は、Initializable#initialize メソッドで利用できます。

リスト 12 に、このビジネス・ロジックのインジェクション部分を示します。

本記事で Guice を選択した唯一の理由は、現時点でもっとも単純かつ簡潔であり、しかも型保証された JSR 330 DI 標準の実装であるためです。 サンプルでは、Guice 実装を InjectorProvider 内にカプセル化しているため、既存のプレゼンタ実装に影響を及ぼすことなく、Weld や OpenWebBeans など、ほかのあらゆる DI フレームワークに容易に置き換えることができます。 DI フレームワークを置き換えた場合に考えられる影響としては、InjectorProvider ユーティリティに調整が生じることと、PersistenceModule が不要になることのみです。

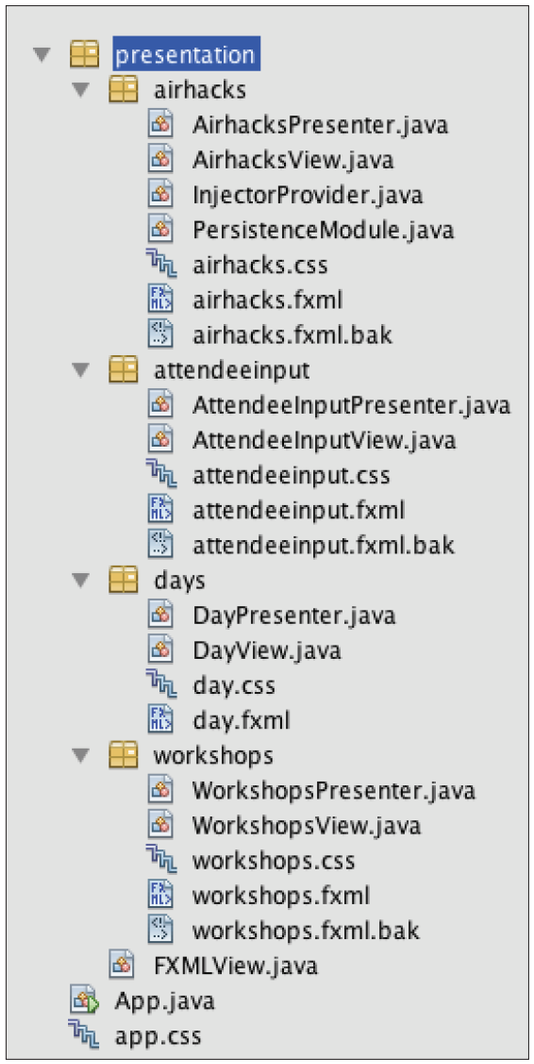


図2

1 つのページに複数のビュー

リッチ・クライアントでは、JavaServer Faces（JSF）で見られるような「粗い」ページ・フローを使用するのではなく、ビュー内部の小さな部分（タブなど）を入れ替えることでフローを実装します。1 つのビューは 1 つのパッケージとして表されます。 あるビューから次のビューへのナビゲーションは、上位のプレゼンタで実装する必要があります。 また、メイン・ビューでアプリケーション・フレームを指定する必要があります。アプリケーション・フレームは、Scene Builder のスケルトンを使用して作成されます。

本項では、前項までと比較して構造に変更はありませんが、命名規則が異なります（**図 2**）。 メイン・ビューのパッケージは、ビューの責務ではなくアプリケーション名を基に命名します。 したがって、アプリケーション・ビューの主な責務は、あるビューから次のビューへのナビゲーションを制御し、共通機能（追加のサポート・クラスなどを格納するアプリケーション・フォルダの保存や読込みなど）を提供することです。

実際、InjectorProvider クラスや PersistenceModule クラスはメイン・アプリケーション・パッケージ内にあります。 アプリケーション・パッケージ（サンプルでは「airhacks」）は、プレゼンテーション・パッケージ内の各ビュー用のパッケージと同じ階層に位置しています。

AirhacksPresenter は、下位のビューを作成して調整し、スケルトン・レイアウトに追加するクラスです（**リスト 13a**、**13b**）。

WorkshopsView は、タブのカプセル化と管理を行い、タブの下にある全領域を置換可能にするためのクラスです。 より正確に言えば、下位のビューは WorkshopsPresenter によって管理されます。この WorkshopsPresenter は、FXMLLoader によって初期化され、WorkshopsView 内にカプセル化されます。

javafx.scene.control.TabPane インスタンスも javafx.scene.control.Tab インスタンスも、

リスト9　リスト10　リスト11　リスト12

```
this.loader.setControllerFactory(new Callback<Class<?>, Object>() {
        @Override
        public Object call(Class<?> p) {
          return InjectorProvider.get().getInstance(p);
        }
      });
```

すべてのリストのテキストをダウンロード

WorkshopsPresenter にはインジェクションされません。 javafx.scene.control.Tab のコンテンツである javafx.scene.control.AnchorPane のみが、WorkshopsPresenter に直接インジェクションされます。 **リスト 14a** と **14b** を参照してください。

Scene Builder では下位のビュー（TabPane ではなく AnchorPane、**図 3** 参照）のインジェクションが可能です。このインジェクションを利用すれば、コードが大幅に簡潔になります。

本記事のケースでは、GRUD は javafx.scene.control.TabPane クラスを参照することなく実現できます。 ただし、タブを変更する操作に応答する必要がある場合は、TabPane をインジェクションすることもできます。Scene Builder 内では fx:id のみを設定する必要があり、プレゼンタ内で TabPane と同じ名前と型のフィールドを宣言する必要があります。

VPV（View-Presenter-View）パターンの登場

FXMLView という 1 つのユーティリティ・クラスを使用することにより、FXML ファイル、ひいては Scene Builder によって生成されたすべてのアーティファクトからの完全な分離が可能になります。 この方法により、再現可能なパターンが登場します。 すなわち、開発者が FXMLView のサブクラスをインスタンス化し、このサブクラスで、対応するプレゼンタのインスタンス化を開始するというパターンです。 FXMLLoader#setControllerFactory メソッドに渡す javafx.util.Callback を実装することで、この初期化プロセスを JSR 330 の実装（サンプルでは Google Guice）に委譲します。 Guice はプレゼンタをインスタンス化するだけではなく、事前に設定されたコンポーネントのインジェクションを実行し、スコープ設定にも配慮します。

サンプルのユースケースでは、RegistrationService のインスタンスを 1 つだけ利用できるようにすることが重要です。 DI フレームワークと統合することで、あらゆるサービスやサービス・ラッパー（Business Delegate パターンなど）が、DI フレームワークによってプレゼンタに透過的にインジェクションされ、インスタンス化されます。

プレゼンタは FXML フレームワークや Scene Builder と緩やかに結合されるのみです。 オプションの @FXML アノテーションと @Inject アノテーションがインジェクション・ポイントを指すだけで、その他の依存性はありません。 また、FMXL を使用しない通常のプレゼンタでもビューがインスタンス化されます。**図 4** を参照してください。

全体として、VPV パターンを使用することで、Scene Builder により自動生成されたコード（XML と Java コードの間に大差はあ

XML が基盤

Scene Builder の大部分は XML に基づいています。 UI 定義は FXML と呼ばれる XML 方言で管理されます。ただし、XML を使用しているものの、重複は発生しません。

リスト13a ／リスト13b ／リスト14a ／リスト14b

```
public class AirhacksPresenter implements Initializable {

   @FXML
   AnchorPane input;
   @FXML
   AnchorPane overview;
   AttendeeInputPresenter attendeeInputPresenter;
   WorkshopsPresenter workshopsPresenter;
   @Inject
   RegistrationService rs;

   @Override
   public void initialize(URL url, ResourceBundle rb) {
      AttendeeInputView inputView = new AttendeeInputView();
      this.attendeeInputPresenter =
(AttendeeInputPresenter) inputView.getPresenter();
      this.attendeeInputPresenter.newAttendeeProperty(
).addListener(new ChangeListener<Attendee>() {
         @Override
         public void changed(ObservableValue<? extends
Attendee> ov, Attendee old, Attendee newAttendee) {
            if (newAttendee != null) {
               workshopsPresenter.newAttendeeProperty(
).set(newAttendee);
            }
         }
      });
      WorkshopsView workshopsView = new WorkshopsView();
      this.workshopsPresenter =
(WorkshopsPresenter) workshopsView.getPresenter();
      this.attendeeInputPresenter.selectedAttendeeProperty(
).bind(workshopsPresenter.selectedAttendeeProperty());
      input.getChildren().add(inputView.getView());
      overview.getChildren().add(workshopsView.getView());
   }
```

すべてのリストのテキストをダウンロード

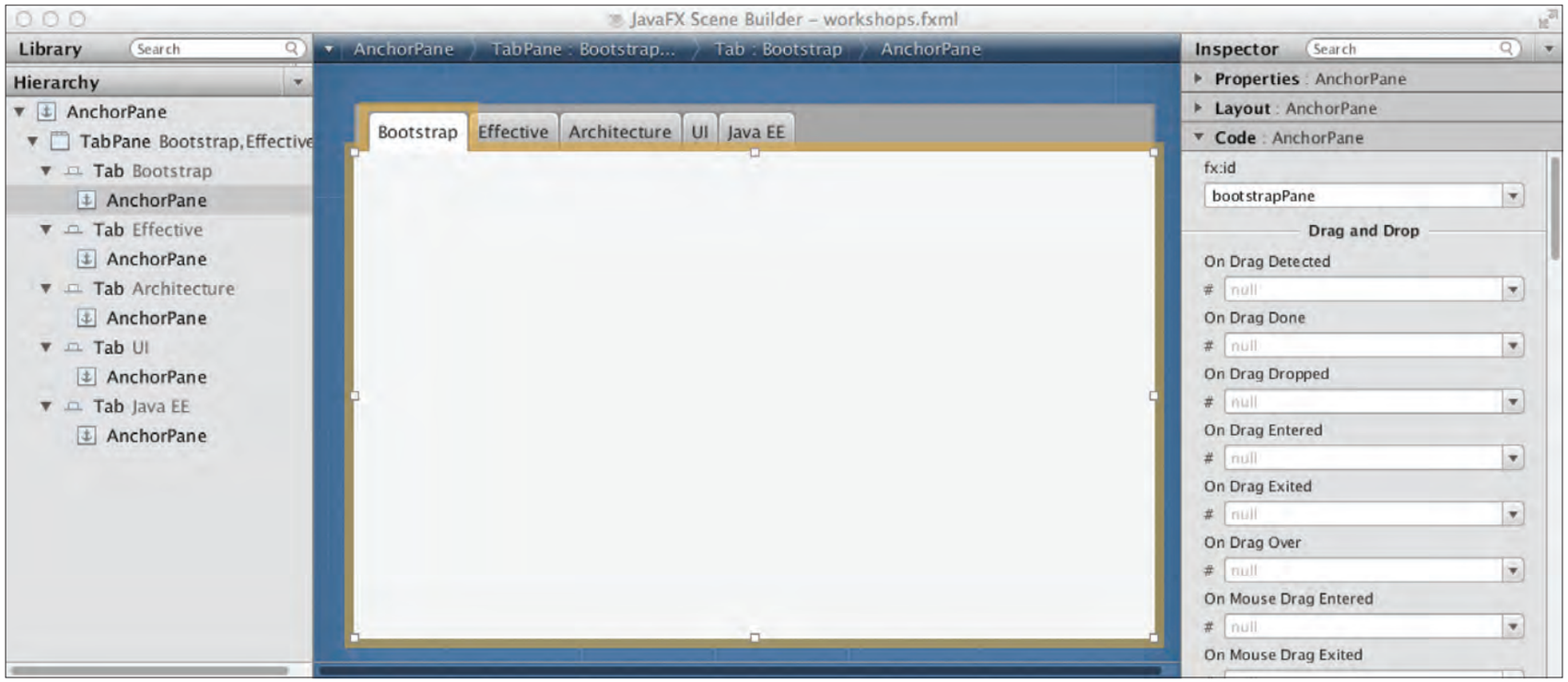


図3

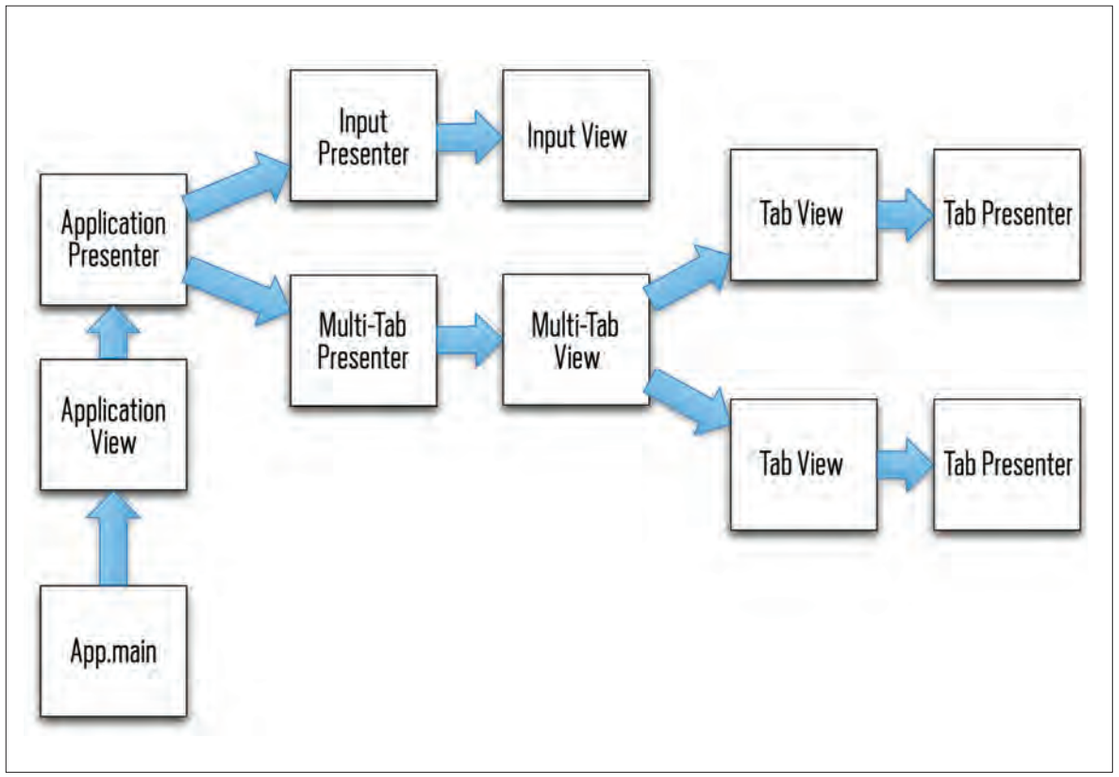


図4

りません）を、手動で作成したアプリケーションにシームレスに統合できます。 サンプルでは、VPV によって、コンポーネント階層がビューから、構成可能なファクトリ（FXMLLoader）に移されました。このファクトリは FXML ファイルによって設定されます。 この FXML ファイルはいつでも、手動でコーディングしたレイアウトに置き換えることができ、プレゼンタを手動でインスタンス化することも随時可能です。

MVP（Model-View-Presenter）パターン

従来の MVP パターンは、2006 年に Martin Fowler 氏により終わりを告げられました。 より正確に言えば、MVP パターンは Passive View パターン、Supervising Controller パターン、Presentation Model パターンに分割されました。

ただし、この 3 つのパターンではデータ・バインディングは考慮されていますが、Fowler 氏は WYSIWYG エディタとの統合には触れていません。 JavaFX では、自動生成されたコードと手動で作成したコードの関係を反転させることで、DI と WYSIWYG を同時に実現します。 本記事で紹介する VPV パターンは極めてシンプルな形式であり、Passive View パターンであると言えます。Fowler 氏は Passive View パターンを以下のように定義しています。

> 「アプリケーション固有の動作のすべてを 1 つのコントローラ内に抽出することで、ウィジェットの状態を完全にコントローラで制御するような画面およびコンポーネントである」

ただし、ビュー・コンポーネントを直接ドメイン・オブジェクトやモデルにバインドすることもできます。このアプローチは、Supervising Controller パターンに定義された目的に似ています。Fowler 氏は Supervising Controller パターンを以下のように定義しています。

> 「UI をビューおよびコントローラに組み込み、ビューでは下層モデルへの単純なマッピングを処理し、コントローラでは入力への応答や複雑なビュー・ロジックを処理する」

厳密に言えば、プレゼンタと JavaFX 固有の要素はそれほど厳格に分離され

てはいません。 プレゼンタは、JavaFX 固有のイベントやコントロールに依存したままです。 しかし、クラスを追加して、簡潔なプレゼンテーション・ロジックのすべてを Java FX に依存しない POJO に配置することもできます。 プレゼンタは、低レベルの JavaFX イベントのすべてを、より高レベルのアプリケーション・イベントに変換します。

この場合、プレゼンタでは、インジェクションされたすべての JavaFX ウィジェットを Java プリミティブとして公開する必要もあります。 一方、「ピュア」な POJO のプレゼンタを使用する場合は、どの UI フレームワークとも結合されないため、Swing、JSF 2、Wicket、Vaadin といった複数のフレームワーク間での再利用も可能です。

JavaFX に依存するプレゼンタは、バックエンドのサービスと同様に容易にモックに置き換えることができるため、プレーンな POJO のプレゼンタを使用することでテストの容易性も向上します。 このようにプレーンな POJO を使用するパターンを Fowler 氏は Presentation Model と呼び、以下のように定義しています。

> 「プレゼンテーションの状態や振る舞いを、インタフェース内で使用する GUI コントロールからは独立した形で表現する」

ただし、こうした定義上、Presentation Model パターンではプレゼンタの数があまりにも多くなります。Presentation Model パターンが MVVM（Model-View-ViewModel）パターンと呼ばれることが多い所以でもあります。つまり、View（FXMLView のサブクラス）が特定の GUI コントロールを保持し、外観を定義します。一方、ViewModel（プレゼンタ）はプレゼンテーションの状態と振る舞いを表します。Model はドメイン・オブジェクト（JPA エンティティの Attendee）になります。 さらに、MVVM パターンでは、コントローラ（AirhacksPresenter のような上位のプレゼンタ）も追加されます。このコントローラで、ワークフローを実装し、複数のビューを仲介します。

しかし、クラスを追加すれば、開発生産性が大幅に低下し、コードの重複も生じます。 ビューに複雑なプレゼンテーション・ロジックが含まれる場合はピュアなプレゼンタを活用することもできますが、極めて単純なビューのために追加のクラスを管理することは良いアイデアとは言えません。

バインドでの関係

Scene Builder では、手動で作成したコードは生成されたアーティファクトに依存しません。依存関係はむしろその逆で、**手動で作成したコードへのバインディングが FXML に含まれます。**この関係は、命名規則に基づいて設定されます。

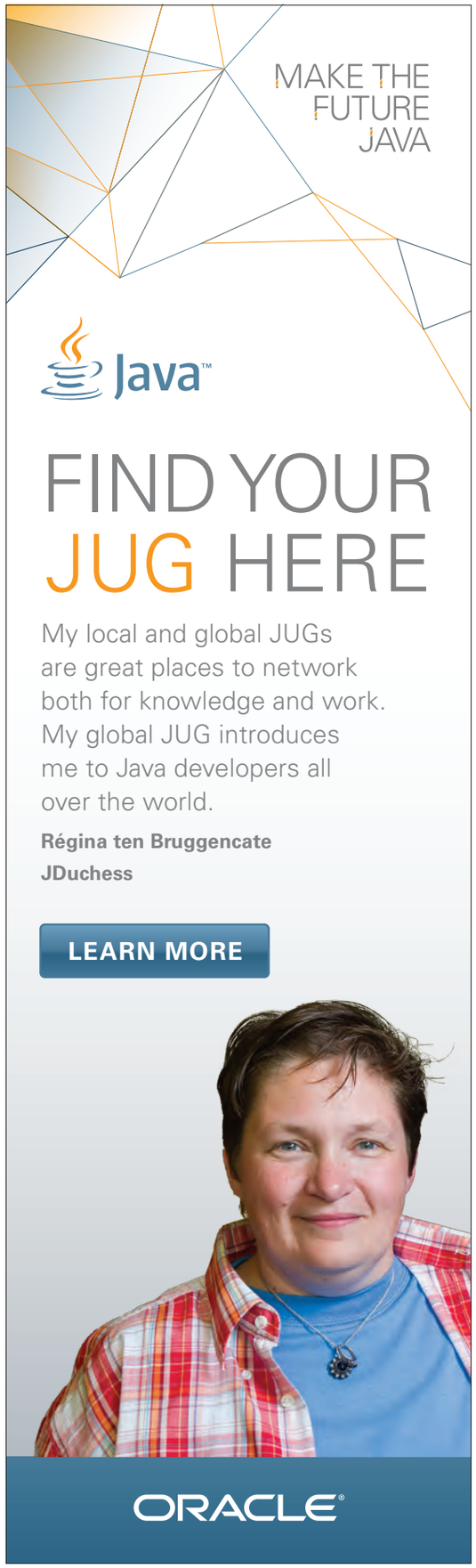
まとめ

自動生成されたコードと手動で作成したコードの関係を反転させることで、迅速なアプリケーション開発を実現できる興味深い機会が得られます。 さらに、「設定より規約」と組み合わせることで、生産性と保守性が同時に改善されます。Scene Builder は、コードの分かりやすさや保守性を損なうことなく使用できます。

このシリーズの次回の記事では、JavaFX データ・バインディングと Scene Builder、FXML、Supervising Controller パターンを組み合わせることで、ビューとプレゼンタの相互作用の詳細を確認します。 **</article>**

LEARN MORE

- Scene Builder
- AirHacks Controlアプリケーションのソース・コード
- AirHacks Controlから「フレームワーク」として抽出したいくつかのアイデア
- afterburner.fx



プログラム可能な通信ネットワーク

オープンな標準ベースのAPIで魅力的な通信サービスを構築する

SEBASTIAN GRABOWSKI、JAROSŁAW LEGIERSKI、DOUGLAS TAIT

BIO

Sebastian Grabowski：Orange Labs Poland のサービス・プラットフォーム / ミドルウェア部門ディレクター。

Jarosław Legierski：Orange Labs Poland のサービス・プラットフォーム / ミドルウェア部門の研究者。

Douglas Tait：オラクルの電気通信部門のマーケティング担当 Global Director。

電気通信は、当初の単純な通話サービスから、動画や、気象情報、交通情報、メディアなどのコンテンツ、その他のインターネットでアクセスできるあらゆる情報を提供する最新型のハイブリッド・サービスに進化を遂げました。 次世代の無線ネットワークはこれらの新サービスに対応しています。 無線ネットワークは、大容量の音声や動画、データを IP 技術のみで処理できるまでに進化しています。 無線通信とインターネット・サービスを組み合わせることは難しいように思えますが、通信業界ではオープンなアプリケーション・プログラミング・インタフェース（API）を通じて中核的なネットワーク資産を公開するというインターネット・モデルを導入してきました。

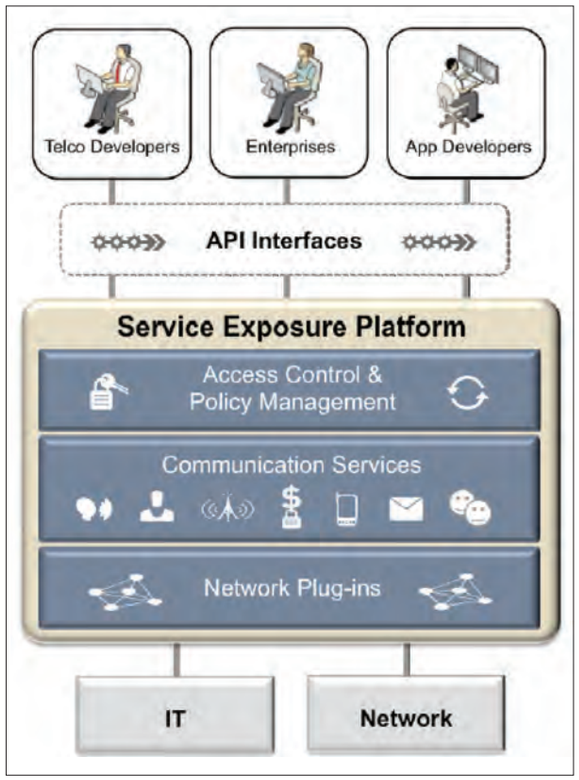


図1

単純な通信 API の例としては、ショート・メッセージ・サービス（SMS）を介したメッセージの送信、ネットワーク上に別のユーザーが存在するかの確認、デバイスの位置情報の取得などが挙げられます。また、電話アカウントへの課金、他者の認証、マルチメディア・メッセージング・サービス（MMS）、マルチメディア会議などの機能がアプリケーションに含まれる場合、通信 API はより複雑なものになります。

本記事では、魅力的で利用者の関心を引き、うまくいけば収益化も可能な通信サービスを構築するための、オープンな標準 API を使用した通信ネットワークのプログラム方法について説明します。

つながりを作るビジネス

通信ネットワークをプログラムするためには、API だけではなく、サービス公開プラットフォームも必要になります。サービス公開プラットフォームとは、インターネットと通信ネットワークを接続し、コネクションサービスプロバイダ（CSP）との関係を確立するためのプラットフォームです。

開発者は、あらゆる API 操作を API サービス公開プラットフォームに対して実行します（**図 1**）。

API サービス公開プラットフォームでは、アクセス制御やポリシー制御を利用して、有害なアクティビティからネットワークを保護します。CSP は、Web サイトやカンファレンス、ニュースレターなどを通じて開発者とのつながりを構築します。構築されたつながりは、アクセス制御やポリシー制御を通じてサービス公開プラットフォームに反映されます

電気通信サービス・プロバイダは、顧客にとって有益な存在であり続けながら、インターネット音声プロバイダとも競争し、同時に自社ネットワークを利用した魅力的なアプリケーションを提供するという難しい課題に直面しています。 単純な通話による収益は減少傾向にあるため、新たなビジネスを確立しようとしています。この新規ビジネスは、各プロバイダの中核的なネットワーク資産を公開して、

このネットワーク上で動作する次世代の優れたアプリケーションを Java 開発者が創出できるようにするというものです。

そのため、電気通信サービス・プロバイダは、自社のビジネスに連携するための標準ベースのインタフェースとなるオープンな API の公開に取り組んでいます。電気通信サービス・プロバイダのビジネスとは言うまでもなく、「つながりを作ること」にあります。この新しい通信サービスは、プログラム可能な通信ネットワークを使用して構築されます。

中核に達する

中核的なネットワーク資産を公開するために、電話通信プロバイダはサービス公開プラットフォームを自社のネットワークに配備します。

通信用のオープンな Java API

インターネット経由で利用でき、多くの業界に対応するオープンな API は、ここ 5 年で飛躍的に発展しました。 たとえば、最新の「ニュース記事」を扱うアプリケーションを開発する場合には、developer.nytimes.com や guardian.co.uk/open-platform で定義されたオープンな API を使用できます。また、Facebook に関するオープンな API については developers.facebook.com、Twitter の場合は dev.twitter.com/doc があります。

ベンダーが API を成功に導く鍵は、自社の最大の強みを明確に示すことです。 電話サービス・プロバイダの場合、その強みは通信機能と請求機能にあります。 一部の電話会社は、ネットワーク API の公開は難しい上に、ビジネスの核となる音声 / メッセージング資産を共有化することにはリスクが伴うと主張します。 通話制御とメッセージングの安全な公開は難しく、そのために API の導入が遅れているのだと言います。 しかし、アクセスとポリシーの保護機能を備え、安全で堅牢であるばかりか重要な収益源ともなるような API サービス公開プラットフォームをすでに展開しているベンダーも存在しているのです。最近になって、複数の電話会社が自社のネットワークを Java 開発者向けに公開するようになりました。 Verizon、AT&T、Orange Labs もその数社に含まれます。

CSP には、API を採用してサービスの技術革新を行ってこなかったという特殊な事情があります。そのために、エコシステムを活用した顧客への価値提供という面で劣勢にありました。 CSP で初めて導入された API は CORBA ベースのものでしたが、CORBA は開発者にとって分かりやすいプロトコルではなく、導入は複雑でした。

その次の世代のオープンな電気通信向け API は、Web サービスとして提供されました。HTML や XML を標準ベースで使用する Web サービス API は、IT 業界で人気がありました。 固定回線かモバイルかを問わず、現在の電気通信ネットワークでは多種多様な Web サービス API が提供されています。ソフトウェア開発者はこれらの API を利用することで、基盤となるネットワークの機能を活用できます。 これらの API は意図的に高レベルの抽象化がなされ、利用しやすいように設計されています。 アプリケーション開発者は、たとえば 1 つの Web サービス要求を呼び出すことで、携帯電話の位置情報を取得や、通話の開始を実行できます。

通信用の標準ベース API の発端は、ParlayX Web Services API および Group Special Mobile Association（GSMA）OneAPI です。

- ParlayX Web サービス：欧州電気通信標準化機構（ETSI）、Parlay Group、Third Generation Partnership Project（3GPP）が共同で定義。 この API は、Web サービス記述言語（WSDL）を使用して定義されており、Web Services Interoperability（WS-I）Basic Profile に従っている。
- OneAPI：GSMA によってサポートされる、インターネット経由でネットワーク機能を公開するための API 一式。 REST インタフェースで構成され、モバイル・ネットワークの各機能（メッセージング、認証、支払など）に世界中からアクセスするために使用できる。

サービス公開プラットフォーム

中核的なネットワーク資産を公開するために、電話通信プロバイダはサービス公開プラットフォームを自社のネットワークに配備します（**図 2**）。

ソフトウェア・アーキテクチャの観点では、サービス公開プラットフォームは、通信事業者のネットワークとインターネットの間に配置されます。

図 2 では、サービス公開プラットフォームの下位（ネットワーク側）のインタフェースが通信事業者のネットワークに接続され、SMS、MMS、位置情報サービス、プレゼンス・サービス、課金サービスなどの電気通信機能と統合されています。 上位（アプリケーション側）のインタフェースは、標準の SOAP インタフェースや REST インタフェースによってインターネットと接続されます。

さらに、サービス公開プラットフォームは、保守やインベントリ、プロビジョニング、障害管理を担う電気通信の運用支援システム（OSS）や、料金設定、課金、請求、支払を処理する業務支援システム（BSS）とも接続されます。

オープンで標準ベースのサービス公開プラットフォームは、API を通じて外部のプログラムと相互に通信し、ポリシーを適用し、サードパーティの開発者やコンテンツ・パートナーによる独自のネットワーク資産も活用できるように設計されています。 多くの電話会社が、自社のネットワーク内にサービス公開プラットフォームを展開して、オープンで標準ベースのサービス公開レイヤーを構築することに成功しています。

電話会社が自社のサービス・ドメインを段階的に発展させて拡張し、IT および Web 開発者のコミュニティの活用を進めるほど、サービス公開プラットフォームを通じて従来の固定回線ネッ

トワークおよびモバイル・ネットワークへの投資（位置情報サービス、支払サービス、メッセージングなど）から得られる収益は大きくなります。 また、IP ネットワーク経由でサービスを公開するための投資は、将来にもつながります。

Java アプリケーション

Orange Labs は、電気通信 API の開発を目的とした Open Middleware 2.0 Community という開発者団体を立ち上げました。

Orange Labs は、複数の大学と協力して、修士課程や学士課程の学生が革新的な通信アプリケーションを開発できるように尽力しました。 300 名を超える Java 開発者によって、プログラム可能なネットワーク API を利用する 30 個のサービスが Java で開発されました。 このコミュニティが開発した興味深いアプリケーションをいくつか紹介します。

- Emergency Button：危険な状態にある人が今どこにいるのかを知るためのシンプルな IT 医療アプリケーション。 このアプリケーションの中心にある考え方は、お年寄りや、健康状態がよくないためにモニタリングを要する人と、その人の状態を知る必要のある保護者などとが、ボタン 1 つで通信できるようにすることです。 このアプリケーションは API を使用して、Unstructured Supplementary Service Data（USSD）メッセージの受信、端末の位置情報の受信、SMS メッセージの送信を行います。
- Telco 2.0 を使用した統合ユニファイド・コミュニケーション：エンタープライズ・ユニファイド・コミュニケーション（UC）システムと、電気通信サービス・プロバイダのサービス配信プラットフォーム（SDP）を実用に耐えるように統合。SMS や USSD によるメッセージ送信、およびユーザーのモバイル端末の地域特定を、UC システムのユーザー・インタフェースから直接実行できます。
- BusStop：都市圏の公共交通機関を支援するアプリケーション。 もっとも近いバス停留所の場所を問い合わせて、ルートや時刻表に関する情報を SMS 経由で受信できます。
- CourierAssistant：宅配会社の輸送計画を改善するアプリケーション。 コールセンターの従業員が、Voice over IP（VoIP）電話で顧客からの注文を受けます。 端末位置検索用の API により、顧客の近くにいる配達員が検索されます。 SMS および USSD チャネルを介して、もっとも近くにいる配達員に、ピックアップの必要な荷物が通知されます。

以降では、ParlayX SMS の導入、OneAPI 支払ゲートウェイの導入、OneAPI 認可アプリケーション、OneAPI QoS API などの事例について説明します。

ParlayX SMS アプリケーションの開発

ParlayX API の利点は、Java 開発者がネットワークの細部をすべて把握していなくても、SMS メッセージの送信などのネットワーク機能を呼び出せることです。 ParlayX は、通信を単純な方法で処理する原始的な手法を提供します。Web ポータルからの SMS メッセージの送信は、サービス・プロバイダが提供する一般的な機能の 1 つです。

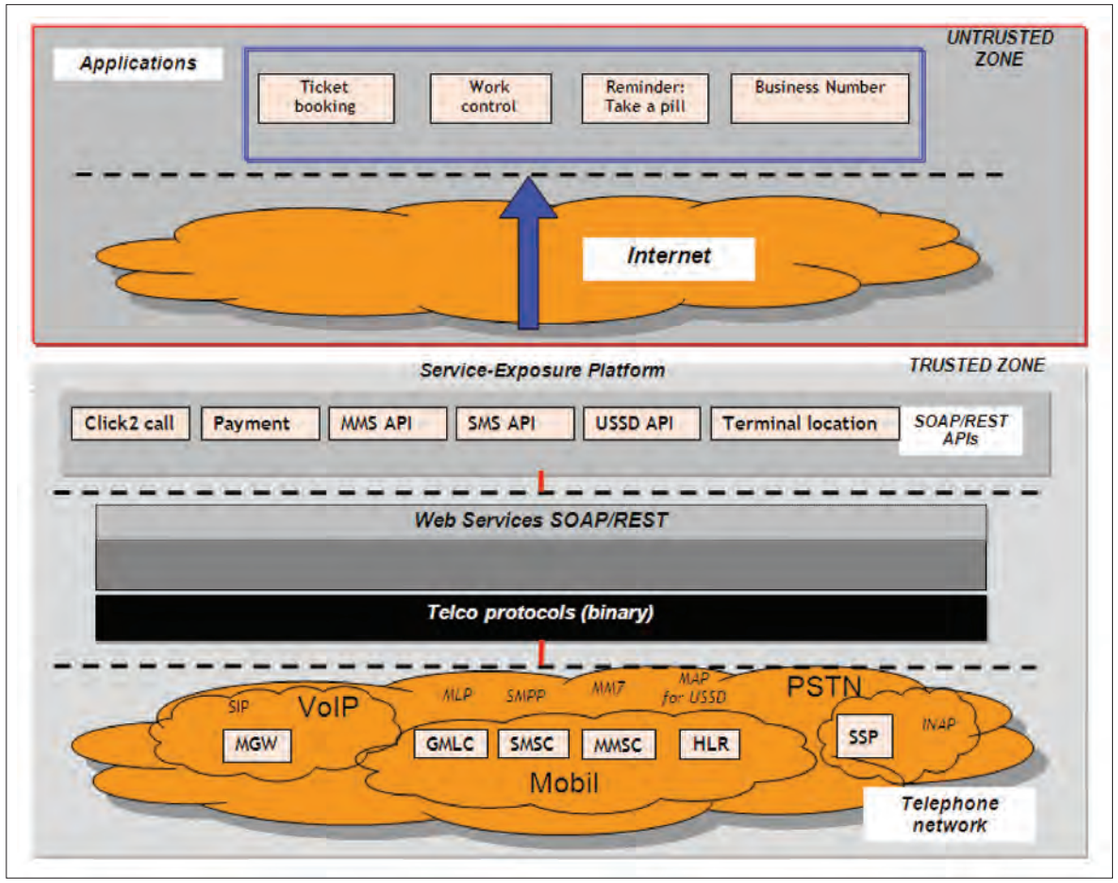


図2

図 3 のシーケンス図に、セルフケア・ポータル・システムが ParlayX SendSMS API 経由で SMS サービスを提供する一連の流れを示します。

SendSms インタフェースのエンドポイントは、http://<host>:<port>/parlayx21/sms/SendSms です。 このアプリケーションのコードは、**リスト 1** のようになります。

SMS メッセージは、registrationIdentifier（Services Gatekeeper のプロビジョニング処理での通知の登録に使用した識別子）を指定して取得します。 **リスト 2** は、Services Gatekeeper により受信されたメッセージを取得するためのサンプル・コードです。

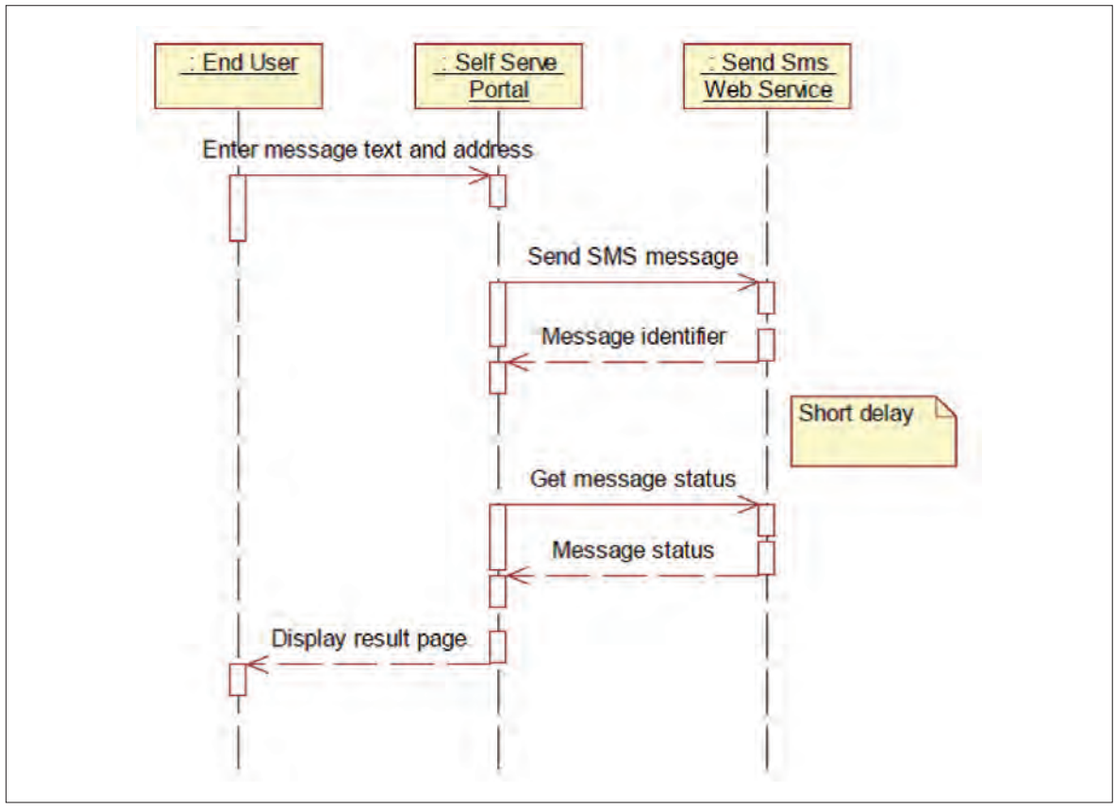


図3

OneAPI Payment API
OneAPI Payment API は、クレジット・カード番号や金融情報、ユーザーのログイン情報、サードパーティの Web サイト経由での登録情報などを収集しなくても購買活動を行えるようにして、顧客エクスペリエンスを向上するための API です。 アプリケーションでこの API を使用すると、エンド・ユーザーのアカウントへの課金、そのアカウントへの返金、課金状況の問合せ、取引の課金額の表示を実行できます。 また、金額の積立、追加金額の積立、積立に対する課金、積立の取消しも行うことができます。 以下に、OneAPI Payment API を使用したサンプルを 2 つ紹介します。

OneAPI アプリケーションの要求は、各種通信サービスで処理されながらサービス公開プラットフォーム内を流れます。 通信サービスは、サービス・タイプ（「マルチメディア・メッセージング」、「端末の位置情報」など）、アプリケーション側のインタフェース（別名上位インタフェース）、およびネットワーク側のインタフェース（別名下位インタフェース）で構成されます。

課金処理のサンプル：課金処理では、Diameter プロトコルを使用して、エンド・ユーザーのアプリケーションに直接課金命令を出します。 **リスト 3** は、課金インタフェースを使用して支払機能を呼び出すための Java サンプル・コードです。

返金処理のサンプル：返金処理では、Diameter プロトコルを使用して、エンド・ユーザーのアプリケーションに通貨額を直接返金します。 **リスト 4** は、支払データ・インタフェースを呼び出すための Java サンプル・コードです。

OneAPI Authorization API
OneAPI Authorization API（OAuth）を使用すると、サービス公開プラットフォームからサードパーティ・アプリケーションのリソースへのアクセスを安全に実行できます。 OAuth では、購読者の認証および購読者のリソースの認可を行い、通信事業者の認証をサービスとしてパートナーのサイトやアプリケーションに公開し、委任認証モードによるサードパーティ認証をサポートし、アクセス・トークンを管理するためのインタフェースを提供します。 以下の 2 つのサンプルにより、OAuth

リスト1 リスト2

```
SendSms request = SendSms();
SimpleReference sr = new SimpleReference();
sr.setEndpoint(new URI(
"http://localhost:8111/SmsNotificationService/services/SmsNoti
fication?WSDL"));
sr.setCorrelator("cor188");
sr.setInterfaceName("InterfaceName");
ChargingInformation charging = new ChargingInformation();
charging.setAmount(new BigDecimal("1.1"));
charging.setCode("test");
charging.setCurrency("USD");
charging.setDescription("some charging info");
sendInf.setCharging(charging);
URI[] uri = new URI[1];
uri[0] = new URI("1234");
request.setAddresses(uri);
request.setCharging(charging);
request.setReceiptRequest(sr);
request.setMessage("we are testing sms!");
request.setSenderName("6001");
SendSmsResponse response = smport.sendSms(request);
String sendresult = response.getResult();
System.out.println("result: " + sendresult);
```

すべてのリストのテキストをダウンロード

の使用方法を説明します。

サンプルの OAuth2 認可処理：リスト 5 では、Client1 がエンドポイント「/oauth2/authorize」への要求を組み立て、サービス公開プラットフォームが Client1 を認証 / 認可（AA）サーバーにリダイレクトします。

サンプルの OAuth2 トークン処理：リスト 6 では、トークン・エンドポイントをクライアントで使用し、受け取った認可情報（つまり認可コード）を提示するかトークンを更新することでアクセス・トークンを取得します。 認証の後、AA サーバーは Client1 を OCSG にリダイレクトして認可コードを発行させ、その後 Client1 が OCSG エンドポイント「/oauth2/a/token」に対する要求を組み立てます。

OneAPI QoS API

顧客が質に応じた支払いを行えるようにするには、サービス公開プラットフォームでサービスを複数の種類に分けて提供します。たとえば、動画の閲覧時には帯域幅を増やす、重要な通話の音声を明瞭にする、ゲームなどのリアルタイム・アプリケーションの場合はレイテンシを抑えるといった対応です。

サービス公開プラットフォームでは、アプリケーションのエントリ・ポイントから QoS の設定を行い、複数のアプリケーション、購読者、ネットワーク・リソース間の関係を管理できます。

以下に、QoS API を使用したサンプルを 2 つ紹介します。

サンプルの ApplyQoS 処理：アプリケーションは ApplyQoS API を使用して、エンド・ユーザー接続に対して一時的な QoS 機能またはデフォルトの QoS 機能を設定することを要求します。 **リスト 7** にサンプル・コードを示します。

サンプルの DeleteQoS 処理：アプリケーションは DeleteQoS API を使用して、エンド・ユーザー接続上で現在アクティブな一時的な QoS 機能を解放します。**リスト 8** にサンプル・コードを示します。

API の推進要因

アクセスとポリシーの保護機能を備え、安全で堅牢であるばかりか重要な収益源ともなるような **API サービス公開プラットフォームをすでに展開している電話会社も存在しているのです。**

リスト3 リスト4 リスト5 リスト6

```
OneAPIClient client = new OneAPIClient(
JAXBContextResolver.class,
"oma.xml.rest.payment._1.ObjectFactory",
"createAmountTransaction", AmountTransaction.class);
client.setUri("http://localhost:8001/oneapi/1/payment/tel%3A1234/
transactions/amount");
client.setAuthorization(getAuthorizationHeader());
AmountTransaction requestData = new AmountTransaction();
PaymentAmount paymentData = new PaymentAmount();
ChargingInformation chargingData = new ChargingInfomation();
chargingData.setAmount(100);
chargingData.setCurrency("USD");
chargingData.setDescription("Sample");
ChargingMetaData metaData = new ChargingMetaData();
metaData.setOnBehalfOf("Example Games Inc");
metaData.setPurchaseCategoryCode("Game");
metadata.setTaxAmount(0);
paymentData.setChargingInformation(chargingData);
paymentData.setChargingMetaData(metaData);
requestData.setTransactionOperationStatus(
TransactionOperationStatus.CHARGED);
requestData.setEndUserId("tel:1234");
requestData.setClientCorrelator("987654321");
requestData.setPaymentAmount(paymentData);
requestData.setReferenceCode("Rfc4006");
try{
  ret = client.post("", requestData);
} catch (final OneAPIClientException e) {
    ……
}
```

すべてのリストのテキストをダウンロード

まとめ

API は今や、企業が中核的な強みを展開して将来につなげるための中心的な戦略となりました。 API は、言わば Web の進化形です。パートナーとの連携を容易にし、オープンな技術革新を可能にし、顧客エクスペリエンスと収益の間の好循環を形成します。

CSP は現在も、自社のネットワークの収益化につながる新しいオープンな API とサービスを求めています。 開発者は、API がオープンであるほどその利点を享受できます。オープンな API は、関心を引くだけでなく収益化も可能なアプリケーションを作成できる唯一の手段であるからです。電話会社にもはや後れをとる余裕はありません。顧客にとって有益な存在であり続けるために、サードパーティだけではなく、社内や既存のパートナーと共同で実施する技術革新を含め、企業全体に及ぶ API 戦略が求められているのです。 **</article>**

ご存じでしたか?

オープンで標準ベースのサービス公開プラットフォームは、API を通じて外部のプログラムと相互に通信し、ポリシーを適用し、サードパーティの開発者やコンテンツ・パートナーによる独自のネットワーク資産も活用できるように設計されています。

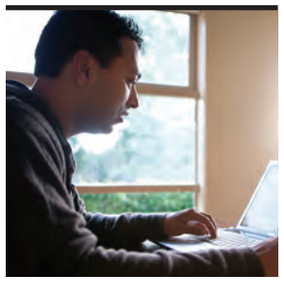
LEARN MORE

- ParlayX Web Services
- OneAPI

リスト7 リスト8

```
ApplyQoSFeatureResponse response = null;
OneAPIClient client = new OneAPIClient(
JAXBContextResolver.class,
"oracle.ocsg.rest.qos.v1_0.ObjectFactory",
"createQoSFeatureProperties", QoSFeatureProperties.class,
ApplyQoSFeatureResponse.class);
QoSFeatureProperties data = new QoSFeatureProperties();
data.setDuration(3600);//unit is seconds
data.setApplicationIdentifier("sampleAppId");
data.setServiceInfoStatus(
ServiceInfoStatus.FINAL_SERVICE_INFORMATION)
data.setsubscriptionId("samepleSubId");
data.setFramedIPAddress(new byte[]{0x0a,0x98,0x47,0x96});
data.setCalledStationId("WAN_1");
MediaComponentDescription mcd = new MediaComponentDescription();
mcd.setMediaComponentNumber(0);
mcd.setMediaType(MediaType.VEDIO);
mcd.setMaxRequestedBandwidthUL(500);
mcd.setMaxRequestedBandwidthDL(1000);
mcd.setMinRequestedBandwidthUL(50);
mcd.setMinRequestedBandwidthDL(100);
mcd.setFlowStatus(FlowStatus.ENABLED);
data.getMediaComponentDescription().add(mcd);
String uri = new String(
"http://localhost:8001/ApplicationQoSService/tel%3A1234/qos");
client.setUri(uri);
try {
   response = (ApplyQoSFeatureResponse)client.post("", data);
} catch (OneAPIClientException e) {
  ......
 }
```

すべてのリストのテキストをダウンロード



VIKRAM GOYAL

BIO

Instagram のようなアプリケーションを Java で構築する方法について、著者 **Vikram Goyal** の解説を聞く

Vikram Goyal：『Pro Java ME MMAPI: Mobile Media API for Java Micro Edition』（Apress、2006 年）の著者。この本では、Java テクノロジー対応電話機にマルチメディア機能を追加する方法について説明。また、『Jakarta Commons Online Bookshelf』（Manning Publications、2005 年）の著者であるほか、手芸品や雑貨の作り方を集めた（Free Craft Projects）Web サイトの管理に貢献。

写真：JONATHAN WOOD/GETTY IMAGES

パート1

Instagramに似た独自のJava MEアプリケーションの構築

画像の撮影、フィルタ適用、変換を行うJava MEアプリケーションを作成する

Instagram が Facebook に約 10 億米ドルもの価格で買収されたというニュースはご存じでしょうか。

このニュースを聞いて、自分でも Java ME ベースのアプリケーションを作成して、もっと成功できるとは思いませんか。 可能性はあります。 Facebook に販売できるかは別として、少なくとも、Instagram アプリケーションに似た動作をするアプリケーションを構築することはできます。

図1

全 2 回シリーズの第 1 回にあたる本記事では、画像の操作と変換の方法について説明します。また、パート 2 では、変換後の画像を Facebook に投稿する方法について紹介します。

注：本記事で紹介するサンプルのソース・コードはこちらからダウンロードできます。

Instagram とは

Instagram とは、iOS デバイスおよび Android デバイスで利用できる写真共有アプリケーションです。これらのデバイスで動作する同種のアプリケーションは多数ありますが、Instagram は使いやすさや画像にレトロ感を出すための豊富なフィルタ、およびソーシャル・ツールへの容易な統合という点で傑出しています。

Instagram アプリケーションの主な構成要素は、フィルタ機能とソーシャル共有機能の 2 つです。この 2 つの機能が、Instagram アプリケーションの成功の鍵となっています。 言うまでもなく、共有機能は Facebook に限定されておらず、Instagram の Web サイト、Twitter、その他の OAuth 認証をサポートする複数の Web サイトでも使用できます。

また、Instagram は独自の API を提供しており、開発者はこの API を使用して Instagram の上層に独自のアプリケーションを作成し、自由にカスタマイズできます。

まずは、Java ME で独自の写真共有アプリケーションを作成してみましょう。

Instagram に似た独自のアプリケーションの作成

Instagram に似たアプリケーションを開発するために必要な処理は、Mobile Media API（JSR 135）を使用した画像の撮影、Image クラスの RGB（赤、緑、青）データを使用した画像の操作、その RGB データのバイト配列への変換です。 いずれの処理も、ごく初歩的なものです。

本記事では説明を簡潔にするために、2 つのフィルタに絞って実装方法を説明します。1 つは「ビンテージ」風フィルタ、もう 1 つはグレースケール（モノクロ）フィルタです。 ただしその前に、アプリケーションのフローと画像操作の仕組みについて理解する必要があります。

PhotoShareMIDlet アプリケーションのフロー

これ以降、PhotoShareMIDlet というシンプルな MIDlet を使用して、画像の撮影、変換、公開というオプションのそれぞれについて一通り確認します。 この MIDlet では、ImageCanvas というクラスを作成し、このクラスを使用して、画像を撮影するためのカメラ（VideoControl）を表示します。 最後に、ImageTransform クラスを使用して画像を実際に操作します。

図 1 は、ある実際の Nokia 製デバイスでアプリケーションを実

行したときのスクリーンショットです。利用可能なすべてのオプションが表示されています。

画像の操作は、ピクセル・レベルで ARGB の各値を変更して行います。ARGB とは、アルファ（Alpha）、赤（Red）、緑（Green）、青（Blue）の略であり、画像データの構成要素です。 先に進む前に、この ARGB の考え方の詳細を理解しましょう。

ARGB について

Java の画像表現における ARGB は、画像のピクセル・レベルの情報を表します。 RGB は、画像の赤色、緑色、青色を表し、A はアルファ・レベルを指しています。 アルファ・レベルとは、各ピクセルの透明度（または不透明度）のことです。 たとえば、アルファ・レベル「0xff」（1111 1111）は完全に不透明であることを表し、アルファ・レベル「0x00」（0000 0000）は完全に透明であることを表します。

ARGB を使用すれば、ある画像を個々のピクセルに分解し、各ピクセルを整数値で表現することができます。 したがって、Java では、画像を単なる整数配列として扱い、ピクセルの ARGB 値を表す整数値をこの配列に格納します。

表 1 は、ある画像の 1 つのピクセルを 32 ビットの整数で表現したものです。

- 先頭の 8 ビット：アルファ・チャネル。画像の特定のピクセルに関する不透明度を表す。 表 1 のピクセルでは完全に不透明
- 次の 8 ビット：赤チャネル。 表 1 のピクセルの赤の値は 0101 0101（0x55）
- 次の 8 ビット：緑チャネル。 表 1 のピクセルの緑の値は 1010 1010（0xAA）
- 最後の 8 ビット：青チャネル。表 1 のピクセルの青の値は 1111 1111（0xFF）

Java ME では画像データをピクセル・レベルで整数配列として扱えるため、データの操作が非常に簡単です。 たとえば、**表 1** の例において、現在の完全に不透明なピクセルを完全に透明なピクセルに変換する必要がある場合は、先頭の 8 ビットを 0000 0000（0x00）に変更します。

Java ME では、Image クラスの getRGB() メソッドを使用して、ARGB データを取得します。 逆に、RGB データを設定して画像を描画する場合は、Image クラスの createRGBImage() を使用します。

以上で ARGB の基本的な理論と Java ME でのデータの操作方法を理解できました。次に、これらの知識を活用して、画像に Instagram のような効果を加える方法について確認していきます。 た

実装の手順

Instagram に似たアプリケーションを開発するために必要な処理：Mobile Media API（JSR 135）を使用した画像の撮影、Image クラスの RGB データを使用した画像の操作、その RGB データのバイト配列への変換

1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1

表1

だしその前に、まずは画像を撮影する必要があります。

画像の撮影

画像の撮影には、Mobile Media API を使用します。Mobile Media API は、使いやすく大変成熟したテクノロジーです（Mobile Media API のみを取り扱った拙著もあります。本記事の最後の「Learn More」をご覧ください）。

本記事のサンプルでは画像を撮影するために、撮影プロセス全体を処理する個別の ImageCanvas というクラスを作成しました。 ImageCanvas のコンストラクタで、VideoControl を初期化しています（**リスト 1**）。これにより、ユーザーがカメラを操作できるようになります。

ユーザーは撮影したいタイミングで「FIRE」ボタン（デフォルトのボタン）を押すことで、スナップショットを撮影できます。 スナップショットは imageArray に格納されて、呼出し側の MIDlet に設定されます（**リスト 2**）。 この処理は別個のスレッド内で実行されることと、画像を撮影してよいか尋ねるメッセージが表示されることに注意してください。 また、この段階で表示されるカメラを別のアプリケーションで使用しないでください。

以上の操作で、PhotoShareMIDlet に 1 つの画像が格納されました。この画像を表示して、編集操作に進むことができます。

画像フィルタの実装

この時点で、ユーザーは前述の 2 つのフィルタ（「ビンテージ」または「グレースケール」）のいずれかを選択して適用できます（**リスト 3**）。 言うまでもなく、これらのフィルタを続けて適用することもできます。

画像の変換には、ImageTransform クラスを使用します。このクラスには、それぞれのフィルタ操作に対応する 2 つのメソッドを実装しています。

ビンテージ・フィルタ：ビンテージ風の外観は、画像を劣化させることで実現します。 昔からの手法として、画像を劣化させるためには、赤チャネルと緑チャネルを変更します。

サンプルでは、2 つの汎用的なメソッドを作成しました。

この「汎用的」とは、赤または緑のチャネルだけでなく、あらゆるチャネルの変更に使用できるという意味です。 1 つ目のメソッドはパブリック・メソッドです（**リスト 4**）。

このメソッドの引数には、各チャネルをどれだけ変更させる必要があるか（変更のデルタ値と呼びます）に関する情報が含まれます。 このメソッドは、元の画像と、各チャネルのデルタ値を引数として受け取ります。 その後、各チャネルの変換に使用するデルタ配列を作成します。

サンプルでは、ビンテージ効果を実現するため、赤と緑のデルタ値をそれぞれ 60 に設定しました。 皆さんも赤と

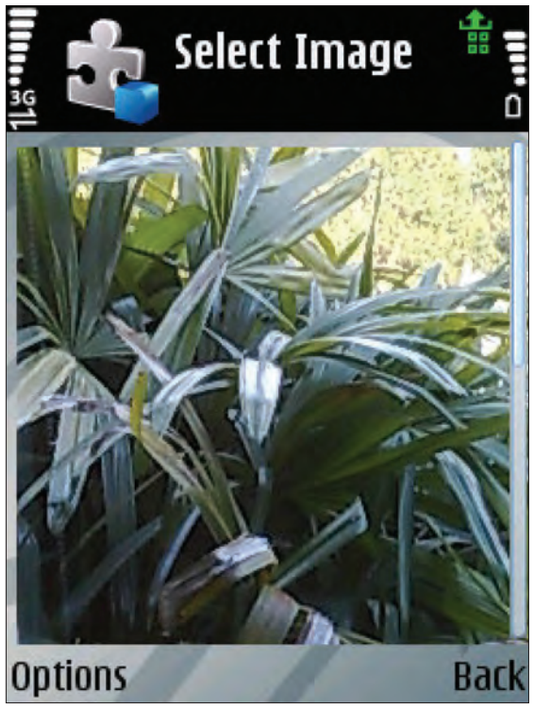


図2

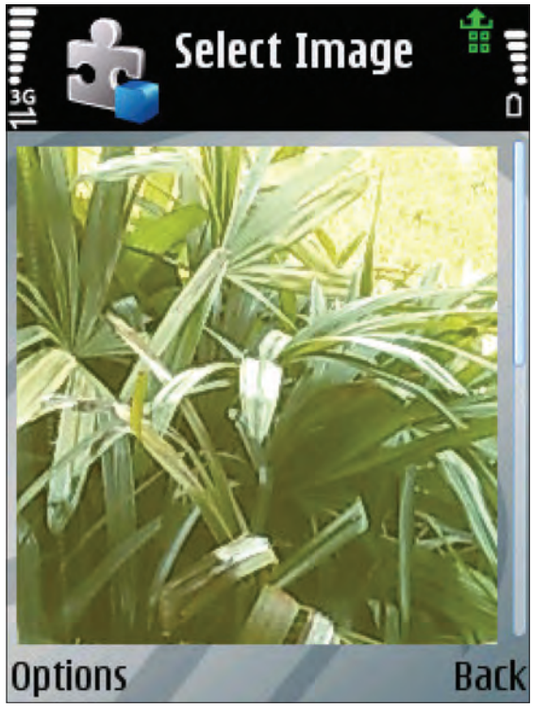


図3

緑（およびアルファ・チャネルと青チャネル）のさまざまなデルタ値を実際に試して、多彩な効果が得られることを確認してください。

このメソッドでは doTransformARGB_Internal という内部メソッドを呼び出し、先ほどの配列の値を引数として渡しています。

この内部メソッドで、整数を実際に操作します。 ビンテージ効果のためには赤と緑のチャネルのみを操作しますが、1つ目のパブリック・メソッドと同じく、この内部メソッドも汎用的なメソッドです。赤チャネルを操作するコードの一部を**リスト5**に示します。

アルファ、緑、青の各チャネルの操作も赤チャネルの場合と似ています（ただし同一ではありません）。それらのコードについては、本記事に関連するソース・コード・ファイルにあります。 **リスト5**のコードにより、メソッドの呼出し時に指定したデルタ配列の値を使用して赤チャネルを変更できます。

図2は元の画像、**図3**は**リスト5**のコードを使用してフィルタを適用した後の画像です。

これらの画像は、実際の Nokia 製デバイスで撮影したものです。

グレースケール・フィルタ：グレースケールへの変換とは、画像からすべての色を取り除いてモノクロ化することを意味します。 サンプルではグレースケールへの変換のために、2つのメソッドを使用しています（**リスト6**）。1つ目のメソッドでは画像内のすべてのピクセルに対して繰り返し処理を実行し、2つ目のメソッドでは色を取り除きます。

リスト1 リスト2 リスト3 リスト4 リスト5

```
public ImageCanvas(PhotoShareMIDlet midlet)
      throws Exception {

  this.midlet = midlet;

  videoControl =
       (VideoControl) midlet.getPlayer().getControl(
       "javax.microedition.media.control.VideoControl");

  if (videoControl == null) {
    throw new Exception("No Video Control for capturing!");
  }

  videoControl.initDisplayMode(
VideoControl.USE_DIRECT_VIDEO, this);

  try { // try and set to full screen
    videoControl.setDisplayFullScreen(true);
  } catch (MediaException me) {

    videoControl.setDisplayLocation(5, 5);
    try {
     videoControl.setDisplaySize(
getWidth() - 10, getHeight() - 10);
    } catch (Exception e) {
    }

    repaint();
  }
  videoControl.setVisible(true);
}
```

すべてのリストのテキストをダウンロード

図4

大ざっぱに言えば、**リスト 6** のコードでは、不透明度はそのままにし、赤、緑、青のすべての平均値を算出することでグレースケールの効果を実現しています。

ビンテージ・フィルタの場合と同様に、**図 2** は元の画像、**図 4** はグレースケール・フィルタを適用した後の画像です。 これらの画像は、実際の Nokia 製デバイスで撮影したものです。

まとめ

この全 2 回シリーズのパート 1 では、Image クラスの高度なピクセル操作を利用してフィルタを作成し、画像を操作する方法について、2 つのフィルタの開発を通じて確認しました。 Instagram に似たアプリケーションを作成する中で、この実装方法を学習しました。

このシリーズの次回の記事では、編集操作後の画像を各自の Facebook ID を使用してオンラインで共有する方法について学習します。

</article>

LEARN MORE

- Mobile Media API
- Pro Java ME MMAPI: Mobile Media API for Java Micro Edition（Apress、2006）

リスト6

```
public static Image doGrayScaleTransform(Image original) {

  int w = original.getWidth(); // original width
  int h = original.getHeight(); // original height
  int[] buffer = new int[w * h]; // buffer for new image

  original.getRGB(buffer, 0, w, 0, 0, w, h); // as RGB values

  // now, for each individual pixel, apply the transformation
  for (int i = 0; i < buffer.length; i++) {
    buffer[i] = doGrayScaleTransform_Internal(buffer[i]);
  }

  return Image.createRGBImage(buffer, w, h, true); // return trans image
}

private static int doGrayScaleTransform_Internal(int bite) {

  // create an int array to hold the 4 levels of
  // transforms we are going to do
  int[] out = new int[4];

  out[0] = (int)((bite & 0xFF000000) >>> 24); // Opacity level
  out[1] = (int)((bite & 0x00FF0000) >>> 16); // Red level
  out[2] = (int)((bite & 0x0000FF00) >>> 8); // Green level
  out[3] = (int)(bite & 0x000000FF); // Blue level
  int new_bite = out[1]/3 + out[2]/3 + out[3]/3;
  out[1] = new_bite;
  out[2] = new_bite;
  out[3] = new_bite;
  return (out[0] << 24 | out[1] << 16 | out[2] << 8 | out[3]);
}
```

すべてのリストのテキストをダウンロード

//fix this /

ヒント：パターン・マッチングについてよく考えること

2013 年 3 月 /4 月号で、Jason Hunter と Boris Shukhat が、Java Persistence API（JPA）によるパラメータ化された SQL とその結果セットの処理方法に関するコードの課題を出しました。 実のところ、コード課題には 2 つのバグが含まれていたため正解は 3 番と 4 番です。

3 番に関して言うと、ネイティブ SQL にパラメータを指定する場合、インデックスは 0 ではなく 1 から始まります。このことは JDBC ドキュメントには記載されていますが、JPA ドキュメントには記載されていません。 プログラマはこれに気づかなかったため、Java のコレクションと同様にゼロから始まるインデックスを使用していました。

4 番についてですが、query.getResultList() を呼び出すと常に List が返されますが、結果セットが空である場合は空の List が返されます。 この事実も JPA ドキュメントには記載されていません。

正しいコードは次のようになります。

```
EntityManager em = ...;
String sql = "select department from emp where fname=? and
mname=? and lname=?";
Query query = em.createNativeQuery(sql);
String[] name=new String[3];
for (int i=1; i<4; i++) query.setParameter(i, name[i]);
List departments = query.getResultList();
if (departments.isEmpty())
    System.out.println("Not available");
else
    for(String department : departments)
        System.out.println(department);
```

今回は、'ポーランドにある Center for Information Technology のソフトウェア開発スペシャリストである Michal Lisiecki からの問題です。

1 問題

ファイルからデータをロードする際、日付の解析で問題が起きる場合があります。 思い浮かぶ簡単な解決策は SimpleDateFormat を使用することですが、これは適切な解決策なのでしょうか。

2 コード

次のコードに目を通して、その結果について考えてください。

```
String value = "01-04-1989";
String dateFormatString = "yyyy-MM-dd";
SimpleDateFormat dateFormat = new SimpleDateFormat(dateFormatString);
try {
   System.out.println("Result: " +
            dateFormat.format(dateFormat.parse(value)));
} catch (ParseException e) {
   System.out.println("Error date parsing: " + value + " in format " +
            dateFormatString);
   e.printStackTrace();
}
```

3 正しい修正方法はどれですか？'

1） Error parsing: 01-04-1989 in format yyyy-MM-dd

2) 0006-09-10

3) 1989-04-01

4) 0104-19-89

わかりましたか？
正解は次号に掲載されます。 もしくはチャレンジした内容を電子メールでお送りください。

画像：I-HUA CHEN