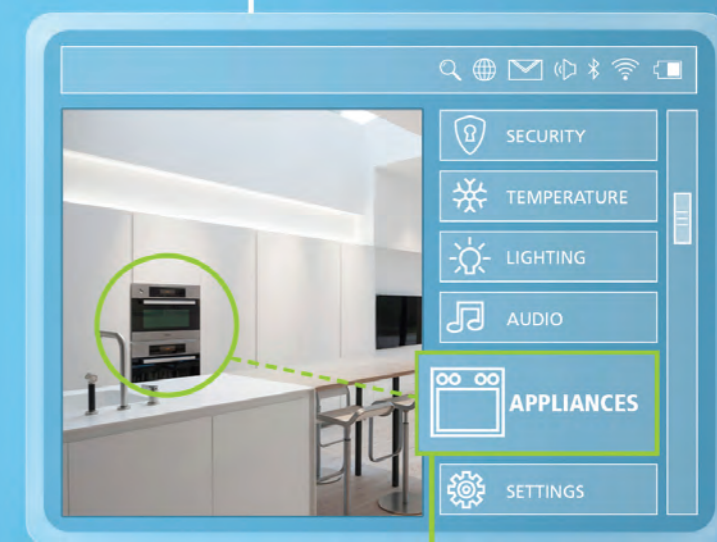


JavaTM magazine

By and for the Java community 

SMARTTEST HOUSE ON THE STREET

～街で一番スマートな家～
Java の openHAB による
Internet of Things



+ 開発の
ベスト・
プラクティス

24 悪魔のパフォーマンス・
チューニング

32 開発者向け DEVOPS

コミュニティ

02 編集長より

03 Java Nation

JavaOne の概要とニュース、人々、イベント

10 JCP Executive シリーズ Patrick Curran 氏との対談

JCP 議長が JCP.next の未来について語ります

JAVA テクノロジー

21 Java アーキテクト アジャイルの手直し

Venkat Subramaniam 博士がアジャイル開発の繊細な問題について考察します

24 Java アーキテクト 悪魔の Java パフォーマンス・チューニング

実践してはならない教訓

26 Java アーキテクト Java Concurrent Animated

もっと優れた並列処理アプリケーションを開発する方法について
Java Champion の Victor Grazi 氏に聞く

32 エンタープライズ Java MySQL、Hudson、Gradle、Maven、Git によるデータベース DevOps

DevOps による、より高品質なソフトウェア開発

39 エンタープライズ Java 開発チームへのメンバーの投入について

新しいチーム・メンバー投入に関する最良、最悪の管理方法

42 エンタープライズ Java Concurrency Utilities for Java EE

アプリケーション・サーバーの並列処理に関するサービスを使用したタスクの最適な実行方法について

54 Fix This

ジェネリクスに関する問題に挑戦しよう



15 SMARTEST HOUSE ON THE STREET

～街で一番スマートな家～
openHAB が Internet of Things とホーム・オートメーションをつなぐ

21

アジャイル、パフォーマンス・チューニング、同時実行性、



DevOps、チーム力学についての記事を含む開発のベスト・プラクティスに関する特別セクション

49 モバイルと組み込み INTERNET OF THINGS 入門

実際に組み込みアプリケーションを作成することで Internet of Things に関連する概念を理解する

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 年代のはじ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。




[あなたが思い描く未来的なプロジェクトについて、是非お知らせください。](#)



編集長 Caroline Kvitka

MAKE THE
FUTURE
JAVA




FIND YOUR
JUG HERE

My local and global JUGs are great places to network both for knowledge and work. My global JUG introduces me to Java developers all over the world.

Régina ten Bruggencate
JDuchess

LEARN MORE



ORACLE®

RamaniはIoTについても触れました。「IoT分野には、オープンな標準プラットフォームが必要であると誰もが考えていますが、Javaはこの市場向けとして最適です。」



オラクルの **Mark Reinhold** は、**James Gosling** 氏の言葉を引用して JavaOne テクニカル基調講演をスタートさせました。Java が成功するには、Gosling 氏のいう "feel of Java" を持続し、信頼性、単純さ、普遍性という重要な価値を維持する必要があります。

Reinhold はこう言いました。「私たちがこれらを維持できれば、Java はその生産性だけでなく、楽しさも持ち続けるでしょう。単純に評判の良い機能を毎年追加するだけでは十分ではありません」。Reinhold は Java における多数のイノベーションへと話を進めました。特に重要な話題について、いくつかご紹介しましょう。

ラムダ式：Reinhold によると、ラムダ式は、単一のものとしては、言語仕様に対する過去最大のアップグレードです。「私たちは今回初めて、JVM（Java 仮想マシン）、言語、ライブラリがともに発展するように入念な調整を行いました。結果はやはり Java らしいものになりました」と Reinhold は述べました。オラクルの **Brian Goetz** が登壇して次のように述べました。「ラムダ式は、今までの Java のプログラミング方法を変えるでしょう。Java はこれまで、データ型を抽象化する優れたツールを提供してきました。私は、動作パターンの抽象化を改善したほうがよいと考えていました。そこで登場するのがラムダ式です」

DukePad : オラクルの **Jasper Potts** と **Richard Bair** は、Raspberry Pi と Oracle Java SE Embedded 8 をベースにした自作組立て式タブレット DukePad のデモを行いました。DukePad は OS として Raspbian Linux を使用し、プラットフォームとして OSGi ベースの JavaFX 環境を使用しています。

上から順に：チェスをする Richard Bair、DukePad を紹介する Jasper Potts、ラムダ式について語る Mark Reinhold。

FREESCALE と INTERNET OF THINGS



Internet of Thingsの展望を語るGeoff Lees氏

Freescal Semiconductor の Geoff Lees 氏は満員の聴衆を前に Java コミュニティ基調講演を始めました。Lees 氏が語ったのは、どのように Internet of Things (IOT) が実現するかという展望についてです。

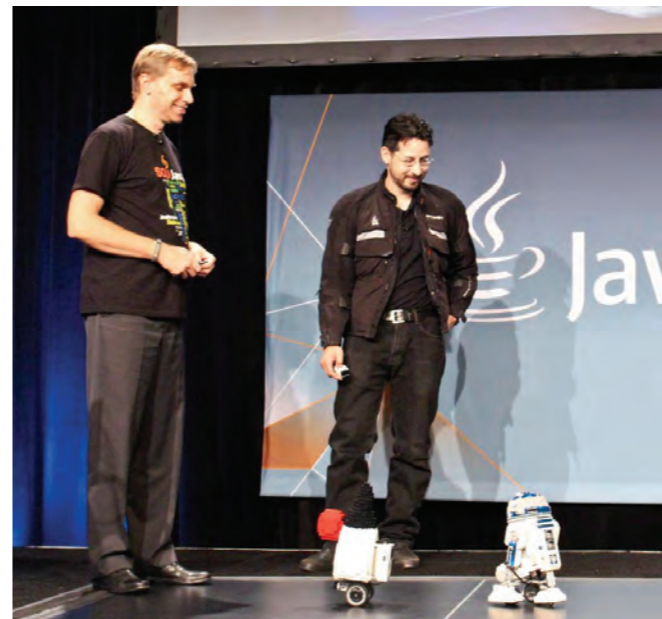
「マイクロコントローラ・コミュニティは急速に Java を採用しており、私たちは助けを必要としています」と Lees 氏は述べました。

Lees 氏によると、IOT は、プロセス・ノードの変化、高度なセンサー・テクノロジーの利用拡大、これらのテクノロジーの統合、プロセスと設計の両技術での低電力テクノロ

ジーの急速な採用といった観点から、半導体業界のテクノロジーに対する考え方を変えつつあります。アナログ信号の集積の進歩と IOT はこれらを実現に近づけています。

「私たちは、今後数年ではなく、今後数か月でこれらすべてをどのように実現するかについて考えています」と Lees 氏は述べ、また、セキュアな IoT を構築するための鍵は Java 開発が握っていると述べました。Java ベースのエッジ・ノードは、ネットワーク全体にセキュアな暗号化サービスと認証サービスを提供する可能性をもたらします。その他の環境でこれらを開発することは困難であり、グローバルに実施することはできないでしょう。また、IoT の開発に必要な転換点には達しないでしょう。

Lees 氏によると、Freescal とオラクルは、エッジ・ノードと各種ゲートウェイ・ソリューションの両方に対応したソフトウェア・モデルおよびハードウェア・モデル向けのプラットフォーム開発に向けて協力しています。「私たちは協業を通じて Java を最適化し、Java 機能をさらにネットワークに取り込むべく努力しています」



コミュニティ 基調講演

オラクルの **Donald Smith** は、Freescal Semiconductor の **Geoff Lees** 氏に続くコミュニティ基調講演で最近の JavaOne カンファレンスを振り返りました。JavaOne 2011 のテーマは Java を前に進めること、そして Java SE 7 リリース後のインフラストラクチャを再起動することでした。JavaOne 2012 では、イノベーションに加えて、クラウドやビッグ・データ、IoT、オープンソースなどの主要技術セグメントにおける Java の役割に重点を置きました。

Smith は続けて次のように言いました。「今年、私たちはこれらすべての一歩先を行き、エンドユーザーやアプリケーション開発者を称えたいと思います。Java エコシステムの懸命の努力のおかげで構築の進んでいる素晴らしいアプリケーションをいくつか紹介しましょう」

この講演中、オラクルの **Henrik Stahl** が壇上で多数のイノベーションが紹介されました。

Java Champion の **Stephan Janssen** 氏は、2013 年の Duke's Choice Award を受賞した Devovx4Kids について語りました。このプ



左から時計回りに：Lego ボットを操作する **Henrik Stahl** と **Stephen Chin** 氏、**Devovx4Kids** について語る **Stephan Janssen** 氏、**Minecraft** プログラミングのスキルについて話す **Aditya Gupta** 君。



ログラムは10歳から14歳までの子供にコンピュータ・プログラミングを教えるものです。Janssen氏は子供向けイベントの開催を考えているJavaユーザー・グループがあれば、Devoxx4Kidsの教材を提供すると申し出ました。

Oracle Academyの**Alison Derbenwick Miller**は、アカデミーが102カ国250万の学生に対して行った普及活動について話しました。

10歳にしてMinecraftの達人である**Aditya Gupta**君は、Minecraftのソース・コードに対して行った拡張（空飛ぶボタン、果てしなく連鎖する爆発、コード遊び）を紹介して聴衆を楽しませました。

2台のDuke Segwayロボットが登場し、Java Championである**Stephen Chin**氏の誘導に従って誇らしげに歩き回りました。

Java Championの**Paul Perrone**氏はJavaで駆動する車を（ビデオで）紹介しました。

Opowerの**Drew Hylbert**氏は、Javaテクノロジーを使用して消費者が節電する方法を紹介しました。

Goldman Sachsのテクノロジー・フェローである**Mike Marzo**氏は、Goldmanの開発者が何年もかけて作成した1億行のJavaコードが持つ価値について語りました。

最後に、"Javaの生みの親"であり、Liquid Roboticsのチーフ・ソフトウェア・アーキテクトである**James Gosling**氏が登壇し、Aditya Gupta君のおかげでGosling氏自身もMinecraftハッカーになりたくなったと感想を述べたあと、Gosling氏はLiquid Roboticsの海洋グライダー・ボットが海から撮影したハワイの風景を紹介し、その仕組みを詳しく説明しました。

コミュニティ基調講演は、Javaがこれまでに成し遂げてきたことと今後の方向性を一新するという自覚と誇りを強く示すものでした。

JAVA.NET アンケート

もっとも重要な JAVAONE トラック

JavaOne開催の数週間前に[Java.net アンケート](#)が行われ、JavaOneで開催される8つのテクノロジー・トラックのうちもっとも重要なトラックはどれかという質問をJavaコミュニティに投げかけました。2週間にわたって行われた調査では、304票の回答が寄せられました。「JavaOne 2013でもっとも重要なトラックは・・・」結果は以下のとおりです。

30%

JavaFXを使用したクライアントと組み込みの開発

21%

すべてのトラックが重要

16%

Javaのセキュリティ

7%

コアJavaプラットフォーム

7%

Java仮想マシンで稼働する新言語

7%

Java EEのWeb Profileとプラットフォーム・テクノロジー

6%

Java Webサービスとクラウド

4%

Javaの開発ツールとテクニック

3%

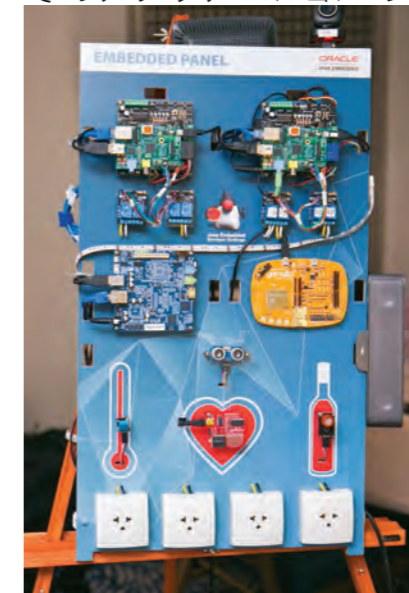
組み込み、スマートカード、IoTアプリケーションでのJavaを使用したエッジ・コンピューティング



JavaOne 前日の土曜
日、大雨にもかかわらず Geek Bike Ride が決
行され、12 名の果敢な
参加者がフィッシャーマ
ンズ・ワーフからゴー
ルデン・ゲート・ブリッ
ジを渡ってサウサリート
までの道のりをサイクリ
ングしました。まさに
Java の粘り強さを物語
るイベントでした。

A man with brown hair and a goatee, wearing a black NASA t-shirt, is standing and speaking at a conference. He is gesturing with his hands. In the background, there is a large window and a chandelier. Other people are visible in the foreground and background, some looking at a screen.

参加者は3日間にわたって、各種センサー、Raspberry Pi、Arduino、その他のボードを統合した Oracle Java SE Embedded アプリケーションを構築しました。チームでのアプリケーションの



左から時計回りに：
Raspberry Pi
の構成を説明
する Vinicius
Senger 氏、
Raspberry
Pi、 Gemalto、
Beagle ボー
ド上のセン
サー、この課
題の目標につ
いて話す Yara
Senger 氏。

開発中、エキスパートとメンターがプレゼンテーションや指導を行いました。

Yara Senger 氏はこう言いました。「一番の目標は達成されました。盛大なパーティ（が開催されていた）にもかかわらず、人々は夜 11 時までコーディングを楽しみ、わずか数日で驚くべきプロジェクトを開発したのです」

開発者チームは、Google Glassを使用したハート・モニター・アプリケーション、ラジコン・カー・アプリケーション、ホーム・オートメーション・プラットフォーム、実地飲酒検査機、電話を使用したリモコン、ロードバランシング・クラウド・アプリケーション、音声 / テキストによる盗難アラーム、という7つのリモコン式アプリケーションを開発しました。

[illegible]

```
sudo apt-get update && sudo apt-get install oracle-java7-jdk
```



左から：Mohamed Taman 氏、Gil Tene 氏（と Heather VanCura 氏）、Brian Goetz 氏

Java Community Process Award

11 年目となる Java Community Process (JCP) Program Award の受賞者が

JavaOne で表彰されました。 カテゴリと受賞者は次のとおりです。

JCP Member/Participant of the Year :

Gil Tene & (Azul Systems)

JCP によると「(Tene 氏は) ソフトウェア特許、知的財産、ライセンス関連の問題に対して明確なアドバイスを提供することで、Java の既得権益を持つ組織だけでなく非常

利団体または個人のためになるよう、熱心に取り組んできました」

Outstanding Spec Lead : Brian Goetz (Oracle)

JCP は、Goetz が「極めて複雑な JSR である JSR 335 (Lambda Expressions for the Java Programming Language) に対して精力的に取り組んでいる」点を評価しました。

Most Significant JSR : JSR 335、 Lambda Expressions for the Java Programming Language

JCP によると、JSR 335 は「Java に対する不満の声を元に、最新のプログラミング言語仕様を取り入れています。・・・」

Outstanding Adopt-a-JSR Participant : Mohamed Taman 氏、 Faissal Boutaounte 氏 (Morocco JUG、EGJUG)

Taman 氏と Boutaounte 氏が評価されたのは、「JSR 339 や JAX-RS 2.0 仕様に
加えて多数の JSR を採用したことです。Morocco JUG が JSR 339 に関して JIRA
に登録した問題の 1 つは 'release-stopper' に分類されています」



JCP Executive シリーズ

未来を描く - JCP.next

透明性および参加状況の改善とオープンソース・プロセスの採用に向けた JCP の発展方針について、オラクルの Patrick Curran が語ります。 **STEVE MELOAN**

写真：BOB ADLER

Patrick Curran は Java Community Process (JCP) 組織の議長を務めています。議長の役割は JCP 内の Program Management Office (PMO) の活動を監督することであり、プロセスと組織の発展、メンバーシップの管理、スベック・リードとエキスパートの指導、Executive Committee 会議の進行、[JCP.org](http://jcp.org) の管理などが含まれます。

Curran は、25 年以上にわたってソフトウェア業界に従事しており、Sun Microsystems とその後のオラクルでの経験は 20 年におよびます。JCP に参加する以前は、Sun のクライアント・ソフトウェア・グループの Java コンフォーマンス・エンジニアリング・チームを率いていました。また、Java の適合性と互換性に関する Sun の方針と戦略の定義を担う Sun Conformance Council の議長でもありました。

Curran は、[W3C](#) (Quality Assurance
ワーキング・グループのメンバーおよ
び Quality Assurance インタレスト・グ
ループの共同議長として参加) や [OASIS](#)
(Test Assertions Guidelines Technical
Committee の共同議長として参加) を含
む複数のコンソーシアムやコミュニティに
積極的に参加しています。

このインタビューでは、JSR プロセスを使用することで、どのように JSR プロセス

プは " 文字どおりの規則 " に従っていますが、これからはその " 精神 " に従う必要があります。公開メーリング・リストを作成しても、ほとんどメールが配信されなかったり、エキスパート・グループ以外のコメントが無視されたりするのであれば、何の意味もありません。また、登録される問題が非常にわずかであるか、登録された問題に基づいて行動が起こされないような公開課題追跡ツールには少しも価値はありません。私たちは、参加を希望する人々に参加方法が周知されており、その参加が効果的になるようにしなくてはなりません。JSR の[ホームページ](#)で、これらすべてを実現するために必要な情報を提供する必要があります。

現在、エキスパート・グループの活動を追跡するレポートの要件をまとめているところです。必要な情報を収集して掲載し、誰でも見られるレポートにする必要があります。こうすることで、参加者の関わりが大幅に向上するものと考えています。

また、PMOは、エキスパート・グループが透明性と参加状況に関する責任をどの程度果たしているのかを私たちが判断できるようにするための情報を提供すべく努力しています。

情報の公開と一般からの圧力に加えて、JSR への投票時に Executive Committee がこの情報を考慮に入れることで、エキスパート・グループがその責任を十分に果たす動機付けとなると私たちは期待しています。

Java Magazine : JCP.next
が、より多くの個人や Java
ユーザー・グループ (JUG)、

その他の団体を JCP プロセスに取り込むにはどのような方法がありますか。

Curran : 公開メーリング・リストとオープンソース開発プロセスの使用を通じて人々の参加を容易にすれば、標準的な Java 開発者にとっても JCP への参加がずっと魅力的なものになるのではないかと考えています。私たちは Adopt-a-JSR というプログラムを通じて JUG の参加を積極的に募っていますが、このプログラムでは開発者が JUG を介して集まり、興味を持った JSR の支援を行います。この活動には、仕様の批評、実装のテスト、文書化などが含まれます。現在のメンバーには 40 以上の JUG が含まれ、合わせて何万人もの開発者の意見を代表しています。JSR 358 では、個人向けの新しいメンバーシップ・クラスを作成に取り組んでいます。ここで手掛けているメンバーシップ合意書は現在の複雑な法的文書よりずっとシンプルで、ときとして進展を阻む雇用者の署名を必要としません。

個人とJUGの募集は非常に良い結果を収めていますが、営利団体の参加を増やすにはさらなる努力が必要です。企業がJCPに参加する一番の動機は活動中

の JSR 開発プロセスであるため、企業の関与を増やすには、多数の JSR を進める必要があります。JSR を開始するのはメンバーであるため、これは Executive Committee や PMO が直接的に左右できることではありませんが、JSR 358 を通じてオープンソース開発プロセスとオープンソース・ライセンスが導入されることで、

すべてをオープンに
「効果的な標準
開発プロセスを
運用するには、
透明性と参加状
況が鍵となります」



企業の参加が大幅に増加すると期待しています。

Java Magazine : JCP.next でこの変革を軌道に乗せるために実施している正の強化と負の強化についてお聞かせください。

Curran：スペック・リードがタイムリーにプロセスを進められるように促す仕組みは多数あります。主要な正の強化は Star Spec Lead プログラムです。このプログラムは毎年の JCP Award を通じて卓越したスペック・リードを公に表彰するものです。JCP Award では、非常に優れた JSR と前年中に大きな貢献をしたメンバーも表彰されます。

負の強化としては、JSR 348 では期限切れの概念が導入されています。特定期間内に定義済みプロセス段階にいたらなかった JSR は期限延長投票にかけられます。このとき、スペック・リードが遅延の正当な理由を説明できず、Executive

Executive Committee メンバーである SouJava の Bruno Souza 氏と話す Curran。



左から：JCP
Executive
Committee 会議
でのプレゼンテー
ションに耳を傾け
る Goldman Sachs
の Mike Marzo 氏、
Curran、Aplix の
Jack Chung 氏。

Committee が建設的な作業の進行を確信できない場合、Executive Committee は JSB の取消しを要求できます。

Java Magazine：仕様、そのリファレンス実装（RI）、および Technology Compatibility Kit（TCK）が同時に完了するように、JCP.next はどのような手法をとる予定ですか。

Curran : PMO は最終承認投票の前に仕様、RI、TCK のすべてが提供可能であることを確認します。ソフトウェア・ライセンスも必要です。これらの要素がすべて整わない限り、投票を進めることはできません。また、次の内容をプロセス文

書に追加しました。14日以内に jcp.org で正式に資料が公開されない場合や、資料に対するリンクが後で切れた場合は、Executive Committee による投票を新たに開始し、スベック・リードが JSR を放棄したものとみなしてこの JSR を取消しに指定できます。これらのチェックを組み合わせることで、プロセス完了時には必要なすべての要素が整い、実装者に継続して提供されることを期待しています。

Java Magazine : JCP.next
によってメンテナンス・プロ
セスにはどのような影響が
ありますか。

Curran : これまでは、メンテナンス・リリースの前に正式な投票が行われていませんでした。RI や TCK だけでなく仕様に対する更新です

ら、公開を義務付ける文言がプロセス文書には含まれていませんでした。ときには、スペック・リードは変更ログを公開するだけであり、新バージョンの採用に必要な作業は実装者自身が突き止める必要があります。これは筋の通らない話です。JCP.next によって最終レビューと同様のメンテナンス・レビューが実施されるようになり、事前に準備された資料を Executive Committee が調査し、一般からもこの資料へアクセスできます。

しかし、他にもやるべきことがあると認識しています。オープンソース・プロジェクトで採用される継続的な開発およびリ

リース・プロセスの種類に合わせてメンテナンス・プロセスの適合性を高めるため、JSR 358 において追加で変更を行っていくつもりです。

Java Magazine : Java SE/EE と Java ME に対する Executive Committee の統合について、詳しく聞かせていただけますか。全体として、陣営とプロセスの両方にどのようなメリットがありますか。

Curran : Java ME は当初、Java SE のサブセットでしたが、時間とともに分化していきました。現在、Java SE の新機能の多く、特に言語機能は Java ME に組み込まれており、これらの類似性と互換性はいつそう高くなっています。ですから、JSR 355 で導入したこの Executive Committee の統合は理にかなうものであり、単純な変更でした。私たちが強調したかったのは Java が 1 つのプラットフォームであるということであり、Java SE/EE と Java ME は時間とともに整合性を高めていくでしょう。2 つの異なる Executive Committee を維持する理由は何ら見つかりませんでした。いずれにしろ、2 つの委員会が別々に開催されたことはほとんどありませんでした。[JSR 360](#) (Connected Limited Device Configuration 8) と [JSR 361](#) (Java ME Embedded Profile) の提出をもって Java SE/EE/ME の統合は進行しています。Java SE/EE プラットフォームと Java ME プラットフォーム間での相乗効果が高まるとともに、メンバーが減ることで Executive Committee が合理化されるものと期待しています。

Java Magazine：JCP 内でのオラクルと他者の勢力バランスはどのようになっていますか。

Curran : これは JCP.next を通じたプロセ



Executive Committee メンバーである TOTVS の David Britto 氏と話す Curran。



スの改定において、もっとも基本的な懸念領域の 1 つです。言うまでもなく、オラクルは Java の幹事およびプラットフォームのスペック・リードとして、また、Java テクノロジーの開発に対する最大の投資家として、特別な役割を担っています。しかし、仮に Java が独占的に所有されていたなら、このような幅広い普遍的な成功を成し遂げることはなかったでしょう。世界中にはおよそ 900 万人の Java 開発者がいます。したがって、プロセスがオープンかつ包括的であるだけでなく、オラクル以外の組織、特に競合他社が協力し、参加できる機会を持つことが不可欠です。

現在、JSR の大多数をオラクルがリード

していますが、これではオラクルの利益になるようにスペック・リードの役割が偏る結果となります。オラクル以外にもエキスパート・グループの活発なメンバーとして参加している組織が確かにありますが、オラクル以外の組織がリードする JSR が増えれば状況はより健全になるでしょう。JSR 358 によってオープンソース開発プロセスとオープンソース・ライセンスが採用されることで、オラクルがリードする JSR により多くの他組織が参加し、さらに、オラクル以外からの JSR の作成が助長されることを希望しています。Java の幹事としてのオラクルの正当な事業利益とオープンで協力的なプロセスを求める声とのバランスを取ることは慎重を要する作業です。

Java Magazine : JCP.next によって透明性と参加状況は明らかに改善されています。最後に、JCP.next によるベスト・プラクティスの推進について詳しく説明していただけませんか。

Curran : 現代において効果的な標準開発組織を維持するには、透明性を高め、一般の参加を可能にすることが不可欠です。

JCP のガバナンスとプロセスがオープンでない限り、人々は参加しようとしません。ここが肝心の点です。このようなベスト・プラクティスは、開発する標準の品質を上げます。活発な参加者が増えれば、より多様な視点が得られ、修正される問題も多くなります。その結果得られるテクノロジーは、広く全体的にメリットをもたらします。

JCP.next は透明性と参加状況を変えるという基本的なことから開始した連続的な取り組みであり、オープンソース開発プロセスとオープンソース・ライセンスの採

用に向けて前進を続ける予定です。非常に活動的で競争の激しい世界で取り組みを続ける上で、これらの変更は Java の強みと妥当性を維持するために役立つでしょう。 </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。

LEARN MORE

- [Patrick Curran のブログ](#)
- [JCP.next](#)



openHAB の Kai Kreuzer 氏
宅の電気キャビネットを確認す
る Kreuzer 氏（左）と Thomas
Eichstädt-Engelen 氏。



SMARTEST HOUSE ON THE STREET

～街で一番スマートな家～

デバイスとアプリケーションを 1 つのまとまりあるネットワークに統合する Java ベースのソフトウェア環境、openHAB が Internet of Things とホーム・オートメーションをつなぐ **DAVID BAUM**

画像：WES ROWELL、写真：TON HENDRIKS

Kai Kreuzer 氏にこっそり近寄るのは容易ではありません。高度に自動化された住宅の玄関に近づくと、センサーが作動してドアの上に取り付けられた Web カメラが動き始めます。呼び鈴を鳴らすと、建物のいたるところにあるスピーカーが iPhone のビデオ・ディスプレイを通じてあなたの存在が警告されます。Kreuzer 氏はリモートからドアの掛け金を外して、あなたを中に入れることができます。さらに数回 iPhone をタップすると、照明の調節、音楽の再生、セン



 **JavaOne 2013 で、Java Magazine の Caroline Kvitka と openHAB について話す Kreuzer 氏と Eichstädt-Engelen 氏。**

カーテンやシャッターなどの一部の機能は複数のカテゴリにまたがります。これらを毎朝自動的に開け、夕方には閉めることで、快適性を維持し、エネルギーを節約できるだけでなく、留守の場合もこれらの動作によって居住者が在宅しているかのように見えるため、セキュリティを強化できます。これらの動作は時間によって自動的に起動することも、住人の在宅を感知するセンサーによって起動することもできます。照明や冷暖房システム、電化製品を自動的に作動させたり、別のシステムを起動させたりすることもできます。たとえば、Kreuzer 氏の洗濯機は洗濯を 1 回終わると、その旨をスピーカーで一斉に通知します。

JAVA コミュニティとの協力

openHAB の最初のバイナリ・ビルドがダウンロード提供されたのは、2010 年の秋でした。それ以来、Eichstädt-Engelen 氏とKreuzer 氏は、Java ユーザー・グループや EclipseCon、Devoxx、JAXconf、

GeeCON などの Java カンファレンスでこのソリューションを紹介してきました。コミュニティは成長を続けており、現在、各種デバイスやアプリケーション、インタフェースへのバインディングや接続を構築する忠実なコントリビュータによって openHAB エコシステムの拡大が後押しされています。

「バイディングはコード部品であり、openHAB から別のシステムへの接続や、総合的な統合を可能にします」と Eichstädt-Engelen 氏はいいます。

現在、Z-Wave、Plugwise、SONOS、Bluetooth、Modbus、EnOcean、KNXを含む商用オートメーション・システムに対して約 50 のバインディングが作成されています。また、openHAB は XMPP、OSGi、Google カレンダーなどのコンソールを提供しています。これらのコンソールとバインディングに加えて、着実に発展しているホーム・オートメーション・テクノロジーのおかげで、照明やその他の電気装置のスイッチを入れるために物理的なボタンを押す必要はなくなりました。壁紙の内部に隠された NFC タグにスマートフォンを向けるだけでスイッチが作動します。または、住人が部屋に入るたびに自動的に電話からデバイスに信号が送られるような openHAB プログラムを作成することもできます。

「もうじき、存在検知
テクノロジーを使用する
ことで、家の中を歩き回

る住人を openHAB で識別できるようにするでしょう」と Kreuzer 氏はいいます。「これによって、時間やその他の任意の変数に応じて、openHAB で照明やシャッターや音楽を人の好みに合わせて調整できるようになります」

エンドユーザーに必要なのは、openHABをダウンロードして、ターゲット・システムの Java ランタイム環境で解凍するだけです。「事実上のワンクリック・インストールです」と Eichstädt-Engelen 氏 といいます。「5 分以内に openHAB システムを実行できるでしょう」

プロジェクトに含まれる openHAB Designer は Eclipse Rich Client Platform アプリケーションであり、openHAB ランタイムを構成するために使用します。これには openHAB 構成ファイルのエディタが付属しており、構文チェック、オートコンプリート、ハイライト表示、コンテンツ支援などの統合開発環境 (IDE) 機能が完全サポートされています。Kreuzer 氏によると、この成熟した開発環境を利用することで自動アクションに対するルールの実装

と展開が容易になります。また、openHABにはスクリプト言語が含まれているため、開発者は新たな自動化ロジックを簡単に定義できます。

「OSGi は openHAB に
 にとって重要な要素です。
 OSGi によって openHAB
 システムにモジュール性
 がもたらされました」と
 Kreuzer 氏は説明します。
 「これにより、コントリ

囲い込みの回避
基本的な設計目標の1つはモジュール性です：簡単にテクノロジーを置換できるため、特定の種類のアプリケーションやデバイスに固定されることはありません。

JAVA がもたらすもの

「Java には膨大なライブラリのエコシステムと優れたデバッグ・ツールがあり、Web では数多くのサポートが提供されているため、本当に便利です」

—openHAB 設立者、Kai Kreuzer 氏

ビュータがモジュールとバインディングの開発をコア開発から切り離して簡単に行えるようになります。ユーザーは該当するモジュールを選び、実行時に openHAB システムに追加できます」

「Java には膨大なライブラリのエコシステムと優れたデバッグ・ツールがあり、Web では数多くのサポートが提供されているため、本当に便利です」と Kreuzer 氏は述べ、続けて次のように言いました。「特定の問題に対する解決策も簡単に見つかります。また、プラットフォームに依存しないため、Linux や Windows、Mac や組み込みプラットフォームのどれで実行することもできます」

Eichstädt-Engelen 氏の見解も同じです。「組み込みシステムに最適な JVM (Java 仮想マシン) は Oracle Java SE Embedded です」

Java を使用することで、他の開発者との協業も容易になります。「別の言語を選んでいたらコントリビュータの数はもっと少なかったでしょう」と Kreuzer 氏は述べ、次のように付け加えました。「私たちは巨

大なエコシステムの一員であるため、ほとんどの場合、必要な答えを見つけられます」

Eichstädt-Engelen 氏と Kreuzer 氏のどちらも openHAB を商用化するつもりはありません。ただし、両氏は最近、Eclipse SmartHome プロジェクトの提案を行っており、これによって openHAB の主要部分が Eclipse ライセンスの下で使えるようになります。この動きは商用製品へ統合するための道を開くものであり、持続可能性を確保するものです。

しかし、openHAB テクノロジーがどこへ向かおうと、オープンソース・コミュニティがどのような進化を遂げようと、両氏は openHAB システムと Java の緊密な連携を維持していくつもりです。

「Java はプラットフォームに依存せず、時代に左右されることもありません」と Kreuzer 氏はまとめます。「Java API のバー



ジョン間での安定性と一貫性は傑出しています。Java の持つ多様性によって、世界中から新しいコントリビュータを簡単に引き込むことができるのです」

openHAB をプログラムし、タブレットを使用して Raspberry Pi 上でプログラムをテストする Kreuzer 氏と Eichstädt-Engelen 氏。

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。



カリフォルニア州サンフランシスコで開催された JavaOne 2013 で、セッション間の休憩中にメールをチェックする Venkat Subramaniam 博士。

Harnessing the Power of Java 8 Lambda Expressions』です。

自らの会社を立ち上げるまでの間、Subramaniam 博士は Halliburton や Raytheon、Invensys などの組織で、プログラマ・アナリストからシステム・アーキテクチャまでのさまざまな職務を経験しています。現在、非常勤教授として働くヒューストン大学のコンピュータ科学の博士号を有する Subramaniam 博士は、2013 年の Java Champion に選出されています。

Java Magazine : アジャイルの終焉がささやかれるようになったのはなぜでしょうか。

Subramaniam 博士 : これは、4 人の目の見えない男が動物園の象を訪れる例え話に少し似ています。男たちは象を見ることはできないのでそれぞれが別の部分に

触った結果、下した結論も異なっていたという話です。

オブジェクト指向開発が企業の役に立つのかという疑いが持たれたときのことを思い出しました。世界が新しいパラダイムに足を踏み入れたとき、成功した企業もあれば苦労した企業もありました。現在、当然の結果としてオブジェクト指向プログラミングは広く受け入れられています。アジャイルに関しても組織は同じような段階、つまり習熟曲線を経ることで、アジャイルが何であるかを把握し、どのように使用するかを理解していくでしょう。

Java Magazine : 組織はどのようにアジャイルを使用しているのでしょうか。

Subramaniam 博士 : 組織がアジャイルに着手した方法は 2 種類あると考えています。一方の組織では経営陣がアジャイルを推進しており、スクラムなどが主流をなしています。もう一方の組織ではボトムアップ方式の取組みが行われており、より技術的な手法や極端なプログラミング手法が優勢になっています。このどちらも理想的とは言えません。なぜなら、組織内には変化を求める声に同調しなかったり、支持しなかったり、または対応を行わない部分があるからです。組織の他の人々が変化を促進する活動に加わらない限り、このような取組みはたいいてい、あまり良い結果につながりません。

アジャイルの採用はアジャイル（機敏）に行わなければなりません。組織は一連の方法を試し、実際に有効な方法を見つけるためにすみ

やかな調整を行う必要があります。アジャイル開発を成功に導くには、組織やチーム、環境がもつ特定の状況に基づいた賢明な手法を適用することが非常に重要です。

Java Magazine : アジャイルの適用に関する問題のうち、企業文化全体にかかわるものはどの程度ありますか。

Subramaniam 博士 : 文化は大きな違いを生みますが、大規模組織には会社全体の文化が存在しないことも少なくありません。私には、ある企業の広々としたオフィスを歩いていて、さまざまな文化に出くわした経験があります。

同じ企業でも、変化や批判、フィードバックへの対応方法がチームごとに異なる場合もあります。一部のチームは他のチームよりも簡単にアジャイル開発に順応するかもしれません。

Java Magazine : アジャイル開発にはあまり向かないタイプの開発者もいますか。

Subramaniam 博士 : チームの作業がサイロ化している場合、アジャイルには適していません。スキルの習得を表すドレイ

ファス・モデルは 5 段階で評価されます。レベル 5 を付けられた人は暗闇に光る逸材であり、他者の助言や介入なしで物事を成し遂げます。レベル 1 にランク付けされるのは初心者であり、放っておくと先に進むのに苦労する可能性があります。実施する活動によって、この 5 段階評価は異なります。つまり、レベル 1 の人もレベル 5 の人というものはなく、活動ごとに違うレベルになるということ

アジャイルの真髄
反復型の増分開発を通じて、協力的かつ巧みにフィードバック・サイクルに適応することがアジャイルの真髄です。

アジャイル・マニフェスト

2001 年 2 月、17 名の開発者がユタ州のリゾート地に集まり、ソフトウェアの最善の開発手法を探りました。その結果として作成された "[アジャイル・ソフトウェア開発のマニフェスト](#)" には次のように述べられています。

「私たちは、ソフトウェア開発の実践あるいは実践を手助けする活動を通じて、よりよい開発方法を見つけだそうとしています。この活動を通じて、私たちは次のことがらを重んじるにいたりしました。

プロセスやツールよりも**個人と対話**を
包括的なドキュメントよりも**動くソフトウェア**を
契約交渉よりも**顧客との協調**を
計画に従うことよりも**変化への対応**を

左側のことがらに価値があることを認めながら、私たちは右側のことがらをより重視します」

さらに、変化や実用的なソフトウェアの重要性から、簡易性、持続可能な開発、対面コミュニケーションの重要性までにわたる 12 の基本的なアジャイル原則が示されました。

です。この調査によればは、どの組織でも平均レベルは 2 になりますが、これは優れているとは言えないレベルです。私たちは積極的な協力とオープンなコミュニケーションを通じて、チームのレベルを上げることができます。

コミュニケーションや建設的な批評をもとと受け入れやすい開発者もいれば、まったく受け入れられない開発者もいるでしょう。後者の場合は問題になります。個人的なプライドとチームワークをうまく両立させるには、慎重にバランスを取る必要があります。エゴとはコレステロールのようなものであり、良い面と悪い面があります。誰もが個人の仕事に誇りを持ちたい一方で、その先にはチームとしての成功がなければなりません。結局は、オープン性と受容性の兼合いに行きつきます。

アジャイル手法がより効果を発揮するのは、自己防衛の手段としてではなく、チームによる発展・向上が見込めるものとして、開発者がアイデアを出すチームです。

Java Magazine：アジャイルが誕生した当初の原則の 1 つは、対面での会話を最善のコミュニケーション形式として重視していました。この方法は個人的な交流と会話の両方を重視しており、これは相対的に対等な関係を意味します。

Subramaniam 博士：ええ、そうです。私には、直接会ったことのない人々や何年か後に初めて会うことになる人々との協業で非常に生産的な関係を築いた経験が多数あります。ですから、良いコミュニケーションを取るために顔を合わせた対話が絶対不可欠であるとは言えません。これは、人々が対話に持ち込む姿勢の問題なのです。2 人の人間が熱心に協業し、共同の成功を重んじる限り、距離がどれだけ離れていようと優れた成果がもたらされるでしょう。私たちが対面コミュニケーションよりもっと重視すべきなのは、互いに築いてきた信頼です。

仮に私が誰かの不満や愚痴をあなたにこぼせば、あなたは自分についてはどう言われているのだろうかという疑問に思うでしょう。これが信頼をむしろ損なう原因です。直接顔を合わせて会話するだけで、これが改善されるわけではありません。相手の姿勢と信頼レベルを素早く確認することが第一歩です。すべての条件が同じになる場合、私は対面での会話がより良い結果を生むと信じています。すべての条件が同じになることはめったにありませんが。

Java Magazine：博士は最近の著書『[Functional Programming in Java](#)：

[Harnessing the Power of Java 8 Lambda Expressions](#)』（Pragmatic Programmers、2013）において、異なる Java プログラミング手法を示唆していますね。これはアジャイル開発にどう関係しますか。

Subramaniam 博士：Java 8 のラムダ式は Java での関数型プログラミングの可能性を開くものです。このプログラミング・スタイルを適切に実施すると、エラーが少なくなり、コードもより洗練された簡潔なものになるでしょう。また、コードの並列化がずっと容易になります。関数形式のコード内に可動部分が少ないということは、自動化テストの作成と表現力の高いコードへの発展が容易になることを意味します。これにより、今までよりも素早くフィードバックを受け入れられるコードにつながります。要するに、アジャイル開発とはフィードバック主導型の開発なのです。反復型の増分開発を通じて、協力的かつ巧みにフィードバック・サイクルに適応することがアジャイルの真髄です。</article>

Timothy Beneke：ジェンダー関連の書籍で知られるフリーランスのライター兼編集者。

LEARN MORE

• [Venkat Subramaniam 博士のブログ](#)



BEN EVANS,
MARTIJN VERBURG

Ben Evans (@kittylust) : 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。

悪魔の Java パフォーマンス・チューニング

実践してはならない教訓

本記事では、パフォーマンス・チューニングにどのように取り組み、すべてのアプリケーションで考慮すべきこの最重要課題にどのように立ち向かうのかについて説明します。

信頼性、保守性、簡潔さ、アーキテクチャの柔軟性、理解しやすさ、あるいは正確性すらも気にすることはありません。重要なのは、すぐに結果が得られることです。問題がないと明日分かったとしても、誰も関心を持ちません。重要なのは、いかなる結果でも、可能な限りすぐに得られることです。誰よりも早く結果が得られれば、何が「正しい」のかを自分で定義できます。賢明な開発者は、何よりもまず実績 (performance) を重視します。

ベンチマークの利用

オンラインで見つけたベンチマークを常に信じましょう。必ずうまく行きます。また、そのベンチマークが実際のアプリケーションのパフォーマンスと常に強く相関すると理解しておくことも重要です（この点は [Wikipedia](#) でも認められています）。

マイクロベンチマークは最高のベンチマークです。楽しく実行できますし、アプリケーションの動作について低レベルの詳細を理解できれば、その後は、上位レベルの動作を低レベルの詳細から単純に推定して、アプリケーション・スタックの残りの部分がどのように動作するのかを推測できます。

マイクロベンチマークが最高である理由は、Java 仮想マシン (JVM) のごく低レベルの部分で、パフォーマンスにわずかな差があると証明できるところにあります。この種の微細な結果は、システム全体へと集約した場合に常に大きな差になります。たとえば、通常の仮想ディスパッチよりも、あるインタフェース・メソッドのディスパッチの方が絶対に時間がかかると、当然言えると思いませんか。

データと結果の処理

DRY (Don't Repeat Yourself、「重複の排除」) 原則は、現代のソフトウェア開発の重要な部分です。パフォーマンスの観点における DRY は、パフォーマンス・テスト

を複数回実行すべきではないという意味になります。開発タスクは他にも非常に多くあるので、パフォーマンス・テストを繰り返すという単調作業に無駄な時間をかけないことが重要です。何と言っても皆さんは非常に優秀なプロフェッショナルです。したがって、テストを完璧にセットアップし、1回目のテスト実行によりすべてのバグを検出しなければなりません。

「統計的有意性」や「分布の形状」などと言い出す人は無視しましょう。そのような人はおそらく、本当に一流のパフォーマンス・エンジニアに求められるすばらしさに対応できない数学オタクに過ぎません。テストは1回実行し、平均の測定のみを行ってください。この測定だけで、煩わしいテスト結果を処理する仕事は完了です。なお、この平均 (average) とは、「統計的平均 (mean)」を意味 (mean) しています。平均を測定した後は、パフォーマンス・エンジニアの本当の仕事である、アプリケーション内のアルゴリズムから、輝きを放つ最後の1滴までパフォーマンスを搾り取る作業

に帰ることができます。

測定作業は、鋭い洞察力を持たない人が行うことです。アプリケーションを開発した皆さんは、どこに問題があるかが正確に分かります。アルゴリズムの最適化は難しい作業です。そのため、最適化こそ本当のパフォーマンス・プロフェッショナルが時間をかけるべきところです。データの収集や分析について気にする必要はありません。

アルゴリズムの最適化

ボトルネックは、そこにあると思う場所にほぼ確実に存在します。自分自身の持つ驚くべき分析スキルを信じましょう。そうすれば、問題の真の原因（通常は自分以外の人が記述したコード）にたどり着けます。

幸いにも、他の開発者のコードを最適化することは簡単です。まずは、最適化されていない単純なアルゴリズムのコードを見つけることです。長時間実行されるループの途中に条件チェックがある部分はすべて、最適化の候補になります。一般的なアルゴリズムを使用するのは愚かです。扱っ

JAVA CONCURRENT ANIMATED

もっと優れた並列処理アプリケーションを開発する方法について Java Champion の **Victor Grazi** 氏に聞く **JANICE J. HEISS**



2009 年、Java 開発者の Victor Grazi 氏は Java Concurrent Animated を初めて公開しました。このアプリケーションは、並列処理を利用した開発を行うためのチュートリアルとして使用できる一連のアニメーションで、多くの称賛を得ています。Grazi 氏は、Java コミュニティへの貢献により、近年、Java Champion に選出されました。現在は、JP Morgan Chase の Securities Lending 部門でバイス・プレジデントを務めています。技術カンファレンスで頻繁に登壇し、彼が一番大切に行っている Java の並列処理や、その他の Java 関連のトピックについて

て語っています。

Java Magazine : Java Concurrent Animated アプリケーションを試した 人はこれまでに何人ほどいますか。

Grazi 氏: 2009 年 7 月に公開して以来、約 20,000 件ダウンロードされました。Java 開発者はおそらく 1,000 万人はいるでしょうから、ダウンロードした方はまだほんの一部です。ダウンロードの多い国は、アメリカ (23%)、インド (14%)、中国 (7%) です。

自己実行型 JAR ファイルはこちらからダウンロードできます。この JAR ファイルをダブルクリックするだけで、アプリケーションが起動します。おもにメニューを使用して操作しますが、上矢印キーと下矢印キーで、さまざま



写真：CHRISTOPHER LANE/GETTY IMAGES

な画像スライドやアニメーション・スライドを切り替えることもできます。このアプリケーションは Windows、Mac、Linux など、すべてのプラットフォームで動作します。実行するためには、Java SE 6 以降が必要です。

Java Magazine : Java
Concurrent Animated に対
する一般的な反響を教えてく
ださい。

Grazi 氏：皆さん、非常に便利だと言えます。多くの方々が本当に楽しんでくれています。特に教師や、チームに対して並列処理の適切なスキルを身に付けてほしいと思っているチーム・リーダーから高い評価をいただいています。Java は、コア・ライブラリに並列処理を取り入れた最初の言語の 1 つです。当時、並列処理は強力な機能でしたが、突如として、非常に優れたプログラマが劣悪なコードを記述するようになりました。適切な並行プログラミングを正しく理解することは難しく、不可能な場合もあります。しかし、時間をとって、そこに潜むフレームワークの一部を理解すれば、並列処理のコーディングでエラーが発生しにくくなるでしょう。

Java のメモリ・モデルを例に挙げると、開発者はしばしばメモリ・モデルを無視して、コードが壊れているとはまったく知らずにコーディングを続けます。そのコードは、Java のメモリ・モデルが実現する最適化を Java 仮想マシン (JVM) やサーバーが活用できない場合があるという意味で壊れています。しかし、メモリのコアが増加し高速化するに従って、メモリのメーカーは、Java のメ

見た目以上の機能

Java Concurrent Animated は単なる Flash アニメーションではなく、 一連の対話型 Java プロ グラムです。各アニメー ションは、説明している 並列処理に関する基本コ ンポーネントを実際に利 用しています。

モリ・モデルのもたらす効率性を利用するようになります。そのため、うまく機能していたコードでも、散発的に障害が発生し始めることになるでしょう。並列処理を正しく管理していないことがその理由です。

Java Magazine : Java

Concurrent Animated は、開発者が Java の並列処理の原則やプロセスをより早く直感的に理解できるように、ある種の「視覚化」を開発者の代わりに行うものなのでしょうか。

Grazi 氏：面白い解釈ですね。Java Concurrent Animated は単なる Flash アニメーションではなく、一連の対話型 Java プログラムです。各アニメーションは、説明している並列処理に関する基本コンポーネントを実際に利用しています。画面の右側にはコードを示すパネルがあります。アニメーションの進行に合わせて、コードの実行箇所が動的にハイライト表示され、実行後には元に戻ります。

ReadWriteLock アニメーションで動作の例を見てみましょう。**ReadWriteLock** は、データ整合性を確保するために使用します。無制限の数の読取りスレッドが読取りロックを取得し、同時に実行できます。しかし、書込みスレッドがロックを取得するためには、すべての読取りスレッドの実行が完了するまで待機する必要があります。書込みスレッドがロックを取得すると、他の読取りスレッドや書込みスレッドはロックを取得できません。

では、動作中の読取りスレッドが読取りロックを解放するまで書込みスレッドが待機中の

ところに、新しい読取りスレッドが登場したとすれば、どちらのスレッドが優先されるのでしょうか。新しい読取りスレッドが書込みスレッドよりも先に進むべきではないか、という考え方があります。つまり、別の読取りスレッドがすでにロックを保持している場合に、新しい読取りスレッドが現在待機中の書込みスレッドがロックを解除するまで、さらに待機しなくてもよいだろう、という考え方です。Java 5では実際に、この考え方に基づいて動作していました。しかし、Java 6でこのアニメーションを実行してみると、動作が以前とは異なることに気づきました。後続の読取りスレッドは、待機中の書込みスレッドのすべてがロッ

利用可能なアニメーション

Java Concurrent Animated で利用できるアニメーションには、Executor (ThreadPoolExecutor、SingleThreadExecutor、CachedThreadPoolExecutor、RejectedExecutionHandler)、Future、synchronized、ReentrantLock、Condition、Semaphore、ReadWriteLock、CountDownLatch、CyclicBarrier、Phaser、AtomicInteger、BlockingQueue、TransferQueue、CompletionService、ConcurrentHashMap、ForkJoinPool があります。

また、Grazi 氏は、Java のメモリ・モデルと Java 8 の新しいプログラミング構造に関するアニメーションも作成しています。

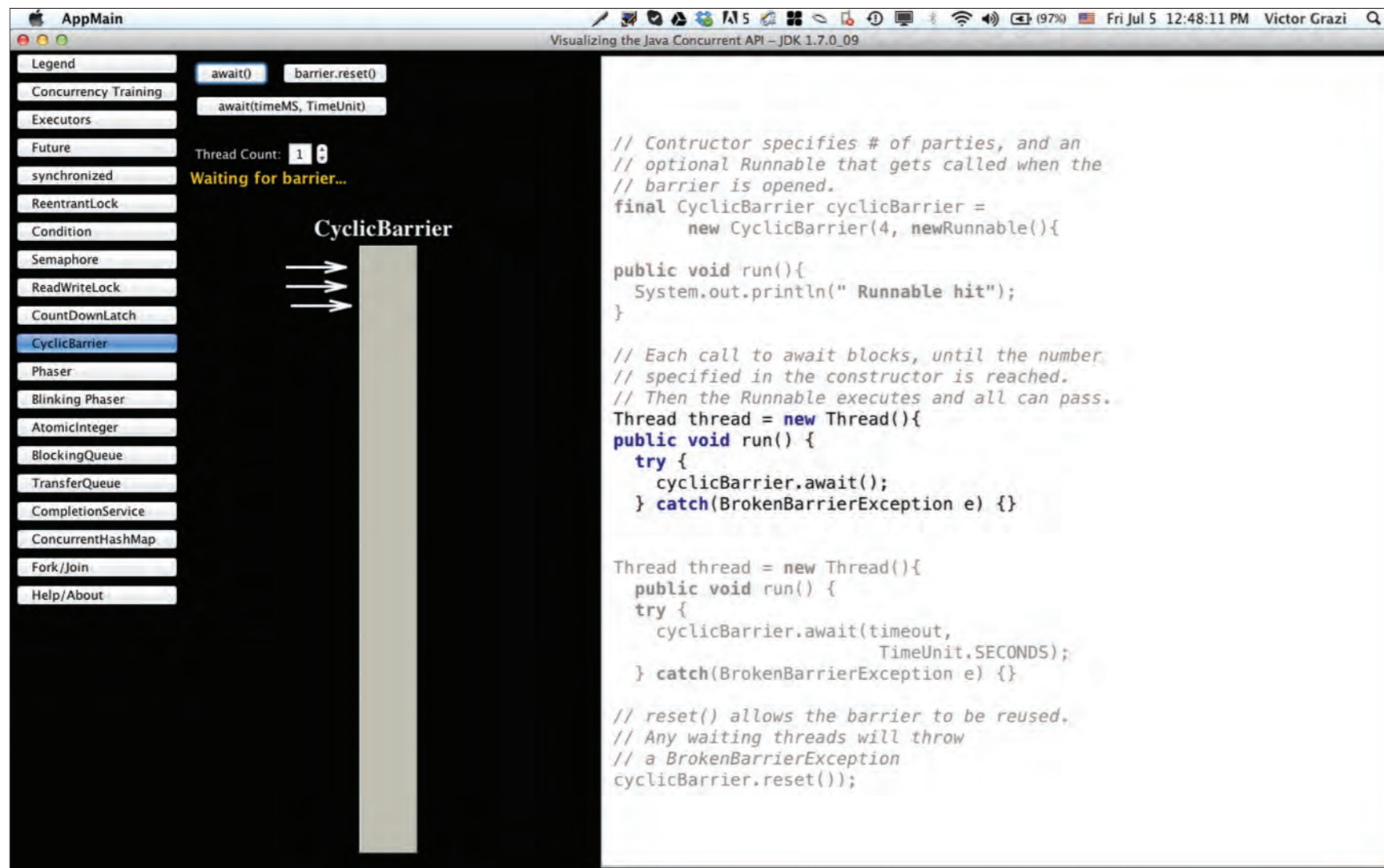


図 2

に多くの処理を実行することを目指して Java が構築されたと言えます。まさに並列処理です。並列処理のバグが発生するリスクを抑えるためのプログラマのスキル向上に、この学習システムはどのように役立ちますか。

Grazi 氏：並列処理の問題に取り組んでいるとき、その解決策がデザイン・パターンにあるとは分かっている、どのデザイン・パターンにあるのかは分からないということがよくあります。Java Concurrent Animated は、開発者が並列処理の問題に対する正しい解決策を探るためのカタログであり、適切な解決

策をもたらす思考の源泉になります。

Java Magazine：Java Concurrent Animated は、Grazi さんが Java の並列処理を利用する開発チームを管理していたときに、ご自身もチームのメンバーもこの機能に対する理解を深められるようにしたいという思いから生まれました。その当初の思いからアニメーションに至るまでの過程について説明していただけますか。

Grazi 氏：私は、並列処理がよく問題になる、投資機関におけるサーバー・サイド Java についてトレーニングを行っています。投資家

は競争相手よりも先に、1 ミリ秒のチャンス を物にして取引を成立させるために、短い 待機時間を求めます。また、バッチ処理がす ぐに完了する必要があるといった要件があり ます。しかし、ぞっとするような書込み専用 のプログラミング構造が見られるようになり、 誰もが並列処理に苦戦していると確信しまし た。私自身、同じように苦戦していました。

ある日の午後、チームに対して並列処理に 関するプレゼンテーションを行うため、空港 でシカゴに向かう飛行機を待っていました。 この間に、PowerPoint プレゼンテーション を仕上げていたのですが、このファイルには 重要なプログラミング構造ごとにまとめられ た一連のスライドが含まれていました。シン プルなテキスト・メッセージを表示する、実 際に動作する小さな状態マシンを記述してい て、これを使用して [java.util.concurrent](#) の並 列処理に関する構造ごとに、重要な状態を 指し示そうとしていました。私は以前に、対 話型ゲームを扱う新興のインターネット企業 で働いていたので、アニメーションにつ いては詳しくかったです。そのため、この PowerPoint のスライドを、よりいっそう直感 的な一連の対話型アニメーションに置き換え られるのではないかと思います。

飛行機を待つ間に最初のアニメーション・ エンジンの開発に取りかかり、その後、先 ほど説明した状態マシンに接合しました。朝 までは、動作するプロトタイプができあが りました。その後数年かけて、このフレーム ワークを強化し、専門家の皆さんからの提 案を取り入れてきました。初期のバージョン を Brian Goetz 氏に送信すると、驚くべきこ とに、それぞれのアニメーションについて助 言をいただけました。この Goetz 氏の提案も すべて取り込みました。さらに、JavaOne で 私が最初に行ったプレゼンテーションに Kirk

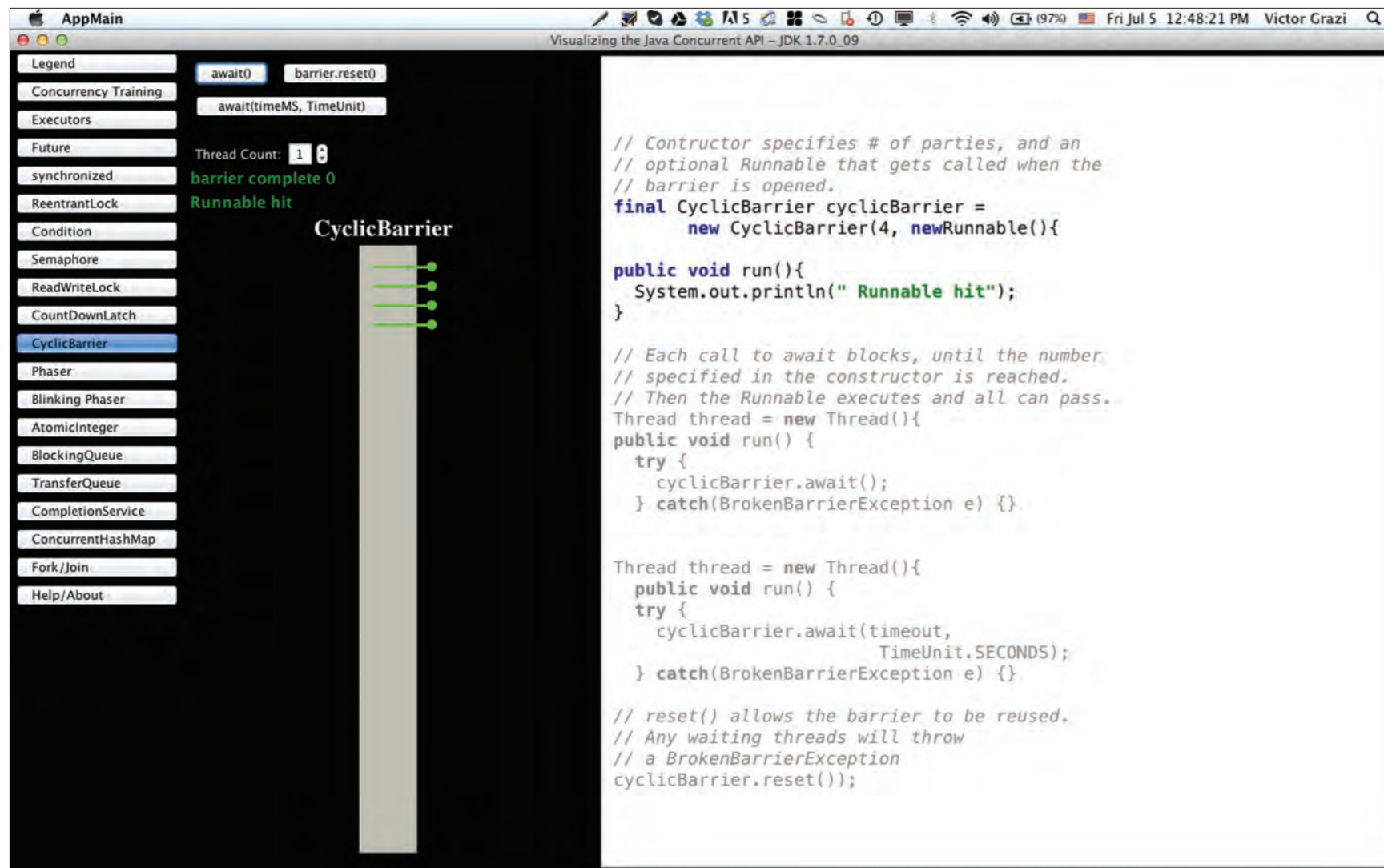


図 3

Pepperdine 氏が参加されました。その際に、発表者が論点を思い出すきっかけとなる PowerPoint の説明をそのままこのアニメーションに追加すればよいのではというご意見をいただきました。説明を追加したことで、発表者だけではなくエンド・ユーザーにとっても本当に便利な機能となりました。さらに、Heinz Kabutz 氏もこのプレゼンテーションに参加し、アニメーションがより直感的なものになるように、いくつかの変更点を提案していただきました。

別のプレゼンテーションでは、情熱的な

ソフトウェア・コンサルタントである Oliver Zeigermann 氏より、[ConcurrentHashMap](#) のアニメーションが存在しないのが気になるという指摘をいただきました。そのアニメーションの提供を Zeigermann 氏に依頼したところ、貴重な追加機能を作成してくださいました。

Java Magazine : Java の並列処理に関するクラスの一部について紹介してください。また、アニメーションにより、開発者は各クラスの理解をどのように深めることができますか。

Grazi 氏 : アニメーションなしで説明するの

は難しいのですが、[CyclicBarrier](#) について見てみましょう。この CyclicBarrier には重要な 2 つの状態があります (図 2 と図 3)。これらの図は、4 つのパーティのバリアを示しています。図 2 では、3 つのパーティしか到着しておらず、これらのパーティは前に進めません。図 3 で 4 つ目のパーティが到着すると、全パーティが前に進みます。

CyclicBarrier はこのように、各パーティはすべてのパーティが到着するまで待機する必要があるというバリアの考え方を示しています。プログラミング構造が複雑化するほど、アニメーションを使用するメリットが大きくなります。たとえば、fork/join のアニメーションや、ネイティブの [wait/notify](#) メカニズムを表すアニメーションなどで、そのメリットを感じられるでしょう。

Java Magazine : アニメーションの作成時に直面した課題について教えてください。

Grazi 氏 : 多くの課題がありました。もともと、このアニメーションではスレッドを矢印で表しており、ほとんどのコンポーネントで問題なく機能しました。しかし、スレッドではなく、[BlockingQueue](#) でキューイングされるオブジェクトを視覚化しなければならないときがありました。この際、「スプライト型」という考え方を取り入れる必要がありました。そして、矢印のスプライト型と、新しい「オブジェクト」のスプライト型を用意して、後者は矢印ではなく長円で表すことにしました。その後、[ConcurrentHashMap](#) と [AtomicInteger](#) で新しいスプライト型が必要になりました。比較とスワップの動作を視覚化しようとしたためです。

さらに、fork/join の説明を追加するときの課題は、既存のフレームワークを使用して、まったく異なる概念をどのように表すかというものでした。また、そこにはもう 1 つの課題



DevOps がソフトウェア開発の品質向上につながる

ソフトウェア開発は難しく、
データベース開発はそれ以上

に難しい作業です。

クリティカル・パスとなるデータベース

データベース開発における日常的な苦勞について具体例を示しましょう。アプリケーション開発者は継続的インテグレーションを利用し、頻繁なチェックインやテストの自動化などを行います。また、ターゲット環境へのビジネス・アプリケーションの継続的デプロイを利用することも多々あります。一方、データベース開発者は一般的に、実際のデータベースのバージョン管理や継続的デプロイに関する基礎知識を持ち合わせていません。

このギャップは、アプリケーション開発 / デプロイとデータベース開発 / デプロイに大きな違いがあることから生まれます。従来、アプリケーション開発にはローカル・ファイルが使用され、ローカルで実施した変更内容はチェックイン時にのみ公開されます。開発者はコードの変更やデバッグをローカルで実施でき、他のチーム・メンバーの作業との干渉が発生することはありません。アプリケーションのデプロイは、ビルド・サーバーからそれぞれの環境に成果物を自動的にコピーすることで実行します。

一方、データベースの開発では通常、一元管理リソースを利用します。しかし多くの場合、ローカルの開発者用データベース、または一元管理データベース内の個別のスキーマを使用して、生産性の高い独立した作業環境を構築します。

さらに、データベースのデプロイは、コピーや置換といった単純なプロセスではありません。たとえば、データベース表を全削除した後新しい構造を持つ表として再作成することはできません。また、データベースのデプロイでは多くの場合、まったく同一のデプロイは2つ存在しません。ソース・データベースとターゲット・データベースのいずれかが、以前のデプロイや新たな開発によって変更または更新されているためです。

DevOps は、データベースの変更管理などを複数の部門にわたって全体的に実施することで、ソフトウェア開発を効率化します。次に、DevOps について詳しく見ていきます。

DevOps の概要

DevOps とは、ソフトウェア・デリバリー・プロセスを効率化するプラクティスを表す言葉です。本番から開発へのフィードバックによる学習と、サイクル時間（開始時点からデリバリーまでの時間）の短縮を重視しています。DevOps はソフトウェアのデリバリーを迅速化するための助力となるだけでなく、個別の要件や基本的な条件との整合性を高めた、より高い品質のソフトウェアの生産にもつながります。

DevOps では、開発と運用の両方で目標、考え方、ツールを一致させることを目指します。DevOps の目的は、開発と運用のコラボレーションを強化することです。その方法として、

目標、考え方、ツールを共通化します。

DevOps によって、組織間の障壁が最小化されます。そのような「1つのチーム」アプローチを採用することで、アジャイル・プラクティスの利用が運用にまで展開されます。開発の専門家も運用の専門家も、緊密に協力してソリューションの「開発」に貢献するという意味で「開発者」になるのです。

DevOps がターゲットとするアクティビティや状況はさまざまです。

非常に多くのアクティビティと状況

DevOps は、次のような非常に多くのアクティビティや状況に対応します。

- カルチャー：プロセスやツールよりも人を重視した考え方をとる。ソフトウェアは、人が人のために作成するものである。
- 自動化：自動化は迅速なフィードバックを得るために DevOps にとって不可欠である。
- 測定：DevOps では測定のための具体的な方法を探る。品質、および共通の（少なくとも調整された）インセンティブが重要である。
- 共有：共有によって、アイデア、知識、経験をやり取りするためのコラボレーション・プ

ラットフォームができる。

データベース DevOps を定義し展開するにあたり、4つの領域を区別することが有効です。図2に、このDevOpsの領域マトリックスの方法について示します。Area 1は、開発を運用へと展開することを表します。データベースの文脈におけるArea 1の一般的なユースケースは、変換スクリプトをバージョン管理システムに配置し、開発と運用で同じデータベース移行ツール（Flyway など）を使用することです。Flyway については後ほど少し取り上げます。

Area 2は、運用を開発へと展開することを表します。データベース DevOps にとって、Area 2は本番システム上のトラフィック（ロックされた行、ブロッキングしているクエリ、リソース競合など）を把握できるようにすることを意味します。

Area 3は、開発を運用に統合することを表します。たとえば、非機能要件に関する制約や共通目標を設定することが挙げられます。共通目標の例として、データベース検索の80%で2秒以内に画面に結果を表示すること（パフォーマンスに関する共通目標）、他のLinuxディストリビューションへの移植が困難になるようなテクノロジーは利用しないこと（移植性に関する

**衝突の理由
開発チームと運用
チームの間ではしば
しば衝突が起こりま
す。そのおもな理由
は目標、プロセス、
ツールが異なること
です。これらの違い
が部門間のギャップ
やサイロ化につなが
ります。**

次に、Maven プロジェクト・オブジェクト・モデル (POM) を見てみましょう。POM は、このプロジェクト向けの Maven ビルド・ツールのメタ情報ファイルです。このファイルに、Flyway の使用方法を定義し、接続先のデータベースに関する情報を記述します。ロジックは Maven プロファイル内に記述します。**リスト 4**に、対応するコードを示します。

次に、Maven 経由で Flyway を起動します。先ほど紹介した専用のプロファイルをアクティブ化する必要があります。この単純な例では、Maven のインストール後に移行自体も開始します。留意すべき点は、Flyway ではクラスパスのスキャンが実行されることです。そのため、移行スクリプトを正しく適用するためには、ターゲット（つまり Maven のターゲット・フォルダ）にそのスクリプトをコピーしておく必要があります。Maven の呼出しは次のようになります。

```
clean install flyway:migrate
-Pdb
```

DEVOPS の役割

DevOps では、開発と運用の両方で目標、考え方、ツールを一致させることを目指します。

この呼出しをもっと簡単にして、洗練された視覚化機能などの他のメリットを得るために、継続的インテグレーション・エンジンである Hudson もデータ

ベース移行に使用します。ビルドは手動で開始できますが、バージョン管理での変更点を Hudson に監視させ、ビルド・スクリプトを呼び出してビルドを自動的に実行することもできます。このスクリプト呼出しにより、情報がコンソールに出力されます (**リスト 5**)。

この例では、データベースは空で、移行が実行されたことはありません。そのため、Flyway でブートストラップ処理（メタ情報の作成、特にスキーマのバージョン番号の作成）が実行され、その後 2 つの移行スクリプトが実行されます（2 つの SQL ファイルをディレクトリに配置しているため）。


データベースを調査すると、Flyway によって新しい表が 2 つ作成されています。

```
mysql> show tables;
+-----+
| Tables_in_mydb |
+-----+
| PERSON          |
| schema_version  |
+-----+
2 rows in set (0.01 sec)
```

1 つは、移行スクリプトによって作成された表です。もう 1 つは、データベースや、データベースの移行に関するメタ情報（データベースの現在のバージョンに関する情報など）が含まれる表です。これらのメタ情報は、特定環境での特定の実行において適用する必要のある文、あるいは

リスト1 リスト2 /リスト3 /リスト4 /リスト5

```
michael@michael-VirtualBox:~/talk/project/devops/src/main/
resources/db/migration$
ls -la
total 16
drwxrwxr-x 2 michael michael 4096 Sep 22 11:16 .
drwxrwxr-x 3 michael michael 4096 Sep 22 11:16 ..
-rw-rw-r-- 1 michael michael 112
Sep 18 07:59 V1__Create_person_table.sql
-rw-rw-r-- 1 michael michael 149
Sep 18 07:28 V2__Insert_persons.sql
```

 すべてのリストのテキストをダウンロード

は適用してはならない文を抽出するために使用します。

次に、PERSON 表を簡単に見てみましょう。この表には 3 行が含まれています (2 つの移行スクリプトで PERSON 表が作成されて 3 つのエントリが挿入されたため)。



図3

また、コマンドラインまたは Hudson から Flyway のレポート機能を実行することで (flywayInfo コマンドを起動して)、移行の現在の状態に関する情報を取得できます。Gradle というビルド・ツールを使用し、gradle flywayInfo を実行して Flyway を起動した場合の出力はリスト6のようになります。

Gradle の動作リスト7に、Gradle のビルド・ファイルを示します。このファイルは、機能的には非常に多くの点で先ほどの Maven POM に対応しています。Gradle を使用すれば、プロジェクトとビルドの情報を Groovy プログラミング言語で記述できます。Groovy は、Java 仮想マシン上で稼働する非常に優れた言語です。このファイルは、Maven POM と同様に、バージョン管理のソース・コード・ツリーの一部にもなってい

```
mysql> select * from PERSON;
+----+-----+
| ID | NAME      |
+----+-----+
| 1  | Peter Meyer |
| 2  | Peter Bonnd |
| 3  | Klara Korn  |
+----+-----+
3 rows in set (0.00 sec)
```

ます。
Flyway では、Maven と Gradle のいずれかから、要件に最適なビルド・ツールを選択できます。
次に、このプロセスに別の移行操作を追加します。3 つ目の移行スクリプトである V3__InsertUpdate_persons.sql は、既存の行の更新とストアド・プロシージャの作成を行い、最後にそのストアド・プロシージャを呼び出して既存の表に新しい行を挿入します (リスト8)。

リスト6 リスト7 リスト8 リスト9

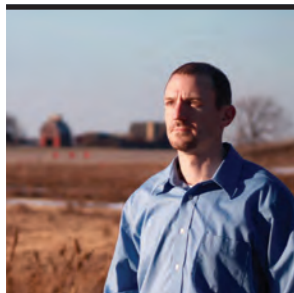
Version	Description	Installed on	State
1	Create person table	2013-09-19 20:52:32	Success
2	Insert persons	2013-09-19 20:52:32	Success
3	InsertUpdate persons	2013-09-19 20:55:52	Success

すべてのリストのテキストをダウンロード

バージョン管理システムの Git にコミットしてプッシュすることで、この新しいアーティファクトを登録できます (リスト9)。
Hudson により Git の変更点が検出され、ビルド・スクリプトの内容に従ってプロジェクトが再ビルドされます。このビルド・スクリプトもバージョン管理のコード・ツリーの一部

です (図3)。
移行フレームワークによってデータベースの現在のバージョンが「2」と検出され、新規に利用できる移行番号「3」を適用するための情報が抽出されて、リスト10のように出力されます。
この時点で PERSON 表に4行目が追加され、さらに更新された行も1

[illegible]



JOSH JUNEAU

Josh Juneau：おもに Java、PL/SQL、Jython を利用する開発者、DBA。Jython Monthly ニュースレターの編集者で、[Jython Web サイト](#)を管理している。Apress 出版の次の書籍の著者または共著者。『The Definitive Guide to Jython』（2010 年）、『PL/SQL Recipes』（2010 年）、『Java 7 Recipes』（2011 年）、『Java EE 7 Recipes』（2013 年）、『Introducing Java EE 7』（2013 年）。

Concurrency Utilities for Java EE

アプリケーション・サーバーの並列処理に関するサービスを使用したタスクの最適な実行方法について

劣悪なパフォーマンスはアプリケーションの成功にとって弊害となります。パフォーマンスはさまざまな方法で測定できます。アプリケーションのユーザーがタスクを開始後、待機せずに別のタスクに移行できる場合でも、ユーザー・インタフェースをブロックせずにデータベースに問い合わせる結果を返す必要がある場合でも、重要なのは、タスクの完了待ちはユーザーにとって非常に不快なもので、アプリケーションのパフォーマンスを低下させて使いにくいものにするということです。

Java SEにおいて、スレッドという考え方は、貧弱なユーザー・エクスペリエンスを防ぐために使用される標準的な技術です。長時間実行タスクを個別のスレッドで生成してバックグラウンドで実行することで、待機が発生しなくなるためです。

Java EE 7に含まれるConcurrency

Utilities for Java EEは、アプリケーション・コンポーネントやJava EEサービスを非同期的に使用するためのソリューションを標準化しています。このAPIは、Java SEの並列処理に精通する開発者が、エンタープライズ・アプリケーションで並列処理を使用するための容易な方法を提供します。

Java EE 6が登場する前まで、サーバーサイド・アプリケーションで複数のタスクを同時に実行することは困難でした。通常、アプリケーション・サーバーのコンテナでは、コンテナにより管理されるスレッド上でアプリケーション・コンポーネントのコードが実行され、さらにその同じスレッド内で、コンテナが提供するオブジェクトへのアクセスも行うためです。アプリケーション・サーバー環境で `java.lang.Thread` や `java.util.Timer` を使用して新しいスレッドを生成することは、信頼できない結果に

つながるため推奨されません。

しかし、Java EE 6で非同期 Enterprise JavaBeans (EJB) が導入され、そのような状況が改善されました。非同期EJBにより、リクエストを処理するスレッド以外のスレッドで、処理を非同期的に実行できます。Concurrency Utilities for Java EEは非同期EJBを基礎として構築されており、一連のアプリケーション・サーバー向けの並列処理を行うための機能を新たに提供しています。並列処理を行うためのサービスは、タスクの非同期実行を補助するものであり、Javaエンタープライズ・アプリケーション用のいっそう充実した並列処理ソリューションを提供することを目指しています。本記事では、並列処理に関する各サービスについて説明し、使用例を示します。

サーバー・リソース

Concurrency Utilities for Java EE
 を利用するために、次のようなア
 プリケーション・サーバーのコン
 テナ・リソースを使用できます。`M
 anagedExecutorService`、`Manage
 dScheduledExecutorService`、`Co`

ntextService、ManagedThreadFactory。これら4つのマネージド・サービス・リソースはアプリケーション・サーバー内に設定され、タスクの非同期実行、コンテナ・コンテキストの取得、マネージド・スレッドの作成に利用されます。

javax.enterprise.concurrent.
ManagedExecutorService
は、渡されたタスクを非同期
的に実行するために使用しま
す。ManagedExecutorService
は、Java SEのJava SE のjava.
util.ExecutorServiceインタ
フェースと同等機能を持つ、
拡張したインタフェースで
す。ManagedExecutorServiceに対
して渡されたタスクは、アプリケー
ション・サーバーが管理するスレッ
ドによって実行されます。

GlassFish 4.xにはデフォルトのManagedExecutorServiceインスタンスがあり、このインスタンスにはconcurrent/defaultManagedExecutorServiceというJNDI名か、Java EE 7 Platform仕様で定義されているjava:comp/DefaultManagedExecutorServiceという標準のJNDI名のいずれ

は、`ManagedTask` インタフェースも実装する必要があります。このインタフェースの実装によって、`IDENTITY_NAME` プロパティ経由でこのクラスに識別可能な名前を付与できます。また、`call()` メソッドも実装する事ができます。このメソッドは、現在設定されているタスクの処理を実行します。このようなタスク・クラスは、非同期的に処理する複数のタスクを発行するために使用します。

長時間実行タスクの送信例を示すために、Acme Worldの支配人が施設の予約状況のサマリーを示すレポートを実行することを望んでいるとしましょう。レポートを起動するために、支配人はレポートを電子メールで受け取るためのボタンをクリックします。このボタンは特定のマネージドBeanアクションにバインドされています。このアクションによって、データベースに問い合わせて予約情報（各予約のコストに関する情報を含む）を取得するための長時間実行タスクが起動されます。このタスクは長時間実行タスクであるため、情報がまもなく電子メールで送信されるということを伝えるサーバー応答をユーザーがすぐに受け取れるようにします。そのため、処理させるタスクを[ManagedExecutorService](#)に渡して、その後に制御をユーザーに戻します。このタスクのロジックを含むクラスの名前を[AcmeReservationReport](#)とし、このクラスのコードを[リスト2a](#)、[2b](#)に示します。

AcmeReservationReportを利用するためには、サーバー上のManagedExecutorServiceリソースに対してAcmeReservationReportを渡す必要があります。これは通常、タスク・クラ

スのインスタンスを作成し、そのインスタンスを[ManagedExecutorService](#)の[submit\(\)](#)メソッドを使用して渡すことで実現します。このメソッドにより[Future](#)オブジェクトが返されます。このオブジェクトを使用して、長時間実行タスクの状態を判定できます。**リスト3**に、処理させるタスク・クラスの受け渡し方法の例を示します。

一部のケースでは、アプリケーションで複数の長時間実行タスクを非同期的に処理する必要があります。その場合は、各タスクの結果が生成された後に処理できます。[java.lang.Callable](#)を実装するタスクがそのようなケースに適しています。

そのようなタスクの例を示すために、週に1回、Acme Worldに予約を行った客についての電子メールを送信することにします。この電子メールは、施設のデータベース表とレストラン予約の表の両方に対して問い合わせた結果を組み合わせることで作成します。そのために、施設の予約を問い合わせるタスクと、レストランの予約を問い合わせるタスクの2つを送信します。これらのタスクが完了したら、まとめて処理して電子メール・レポートの内容を組み立てます。

リスト4に `AcmeParkReservation` タスクのソース・コードを、**リスト5**に `AcmeRestaurantReservation` タスクのソース・コードを示します。各タスクは、`Callable` インタフェースと `ManagedTask` インタフェースを実装しています。コンストラクタはタスクの実行に必要な引数を受け取ります。また、`call()` メソッドでは長時間実行タスクの実装が実行されます。この例では、各

リスト1

リスト2a

/リスト2b

リスト3

```
<resource-env-ref>
  <description>
    Reference to ManagedExecutorService
  </description>
  <resource-env-ref-name>
    concurrent/__defaultManagedExecutorService
  </resource-env-ref-name>
  <resource-env-ref-type>
    javax.enterprise.concurrent.ManagedExecutorService
  </resource-env-ref-type>
</resource-env-ref>
```

 すべてのリストのテキストをダウンロード

タスク内でデータベースに問い合わせ、その後の処理に使用するための情報が返されます。

JavaServer Faces (JSF) マネージドBeanコントローラの `AcmeSummaryEmailController` は、2つのタスクのそれぞれを処理して、各結果を組み合わせて電子メール・レポートを組み立てるために使用します。レポートを構築するために、各タスクを個別に `ManagedExecutorService` に渡して処理させます。タスクごとに `Future<Object>` が返されます。この `Future` オブジェクトを使用し、オブジェクトの `isDone()` メソッドを呼び出して、タスクが完了したかどうかを判定できます。タスクの完了後は、このオブジェクトの `get()` メソッドを呼び出して返されるオブジェクトから情報を取得できます。**リスト6a、6b** は、`AcmeSummaryEmailController` JSFコントローラのソース・コードです。

ManagedExecutorServiceインスタンスは、複数のアプリケーション・コンポーネントから同時に使用できるインスタンスです。各タスクはコンテキスト依存型になります。つまり、各タスクで送信コンポーネントのコンテキストが維持されます。そのタスクは、送信コンポーネントのコンテキスト内で実行されます。ManagedExecutorServiceインスタンスは、サーバー自体のシャットダウン時、またはアプリケーションやコンポーネントの無効化/削除時に、終了または一時停止することが可能です。

スケジュール管理されたタスク

javax.enterprise.concurrent.
ManagedScheduledExecutorService

は、指定した時刻にタスクを実行するために使用できます。`ManagedScheduledExecutorService`とのインタフェースは、すべてのタスク・クラスで`java.lang.Runnable`インタフェースまたは`java.util.concurrent.Callable`インタフェースのどちらかを実装する必要があるという点において、`ManagedExecutorService`の操作と非常によく似ています。並列処理に関するこれら2種類のリソースには、唯一、スケジュール管理されたタスクを渡すときにスケジュール機能を指定できるかどうかという点に大きな違いがあります。

リスト7にAcmeReservationCountというタスク・クラスを示します。このタスク・クラスは、Acme World予約データベースを定期的にポーリングして、新しい予約数を把握するために使用します。この実装にはスケジュール情報は一切含まれず、タスクの実装のみが含まれている点に注意してください。

前述のとおり、スケジュール設定はタスク受け渡し時に実行します。**リスト 8**に、`AcmeReservationCount`クラスを`ManagedScheduledExecutorService`に渡して処理させる方法の例を示します。この例では、このタスクが60分おきに実行され、過去60分間の新しい予約数がログに記録されます。このスケジュール設定のために、`ManagedScheduledExecutorService`のいずれかのメソッドを呼び出しています。これらのメソッドは、`java.util.concurrent`のインタフェースから継承されたものです。

この例では、`scheduleAtFixedRate`メソッドを呼び出しています。このメソッドは、`Runnable`、`initialDelay`、期間、期間の

リスト4

リスト5

リスト6a

リスト6b

注:このリストは、紙面の都合上抜粋になっています。省略部分は...記号で示しています。コード・リストの全体は、本号のコード・リストをダウンロードしてご確認ください。

```
public class AcmeParkReservation implements Callable<
    ParkReservation>, ManagedTask {
```

```
String reportId;  
BigDecimal parkReservationId;  
Map<String, String> executionProperties;  
private ParkReservationFacade parkReservationFacade;
```

```
public AcmeParkReservation(String reportId,  
BigDecimal parkReservationId,  
ParkReservationFacade parkReservationFacade){  
    this.reportId = reportId;  
    this.parkReservationId = parkReservationId;  
    this.parkReservationFacade = parkReservationFacade;
```

```
executionProperties = new HashMap<>();
executionProperties.put(
    ManagedTask.IDENTITY_NAME, getIdentityName());
}
```

```
public String getIdentityName() {
    return "AcmeParkReservation: reportId=" +
reportId + ", parkReservationId=" + parkReservationId;
}
```

```
@Override
public ParkReservation call() throws Exception {
    // Perform long-running task
    return parkReservationFacade.findById(parkReservationId);
}
```



 すべてのリストのテキストをダウンロード

単位を引数として受け取ります。この例では、期間の単位は`TimeUnit.MINUTES`です。スケジュール管理されたマネージド・タスクは、`AcmeWorldServletContextListener`というサブレット・コンテキスト・リスナー内に実装しています。アプリケーションの起動時にこのコンテキストが初期化され、1時間おきにタスクが実行されるようにスケジュール設定されます。

ManagedScheduledExecutorServiceインスタンスは、複数のアプリケーションやコンポーネントから同時に利用できます。このサービスに渡された各タスクはコンテキスト依存型です。また、このタスクを実行するスレッドのコンテキストは、実行中のタスク・インスタンスのコンテキストに一致するように変更されます。タスクの完了後、コンテキストは元の状態に戻ります。

コンテキスト依存オブジェクトの作成

javax.enterprise.concurrent.

ContextServiceインタフェースは、マネージド・エグゼキュータを使用せずにコンテキスト依存オブジェクトを作成するために使用できます。そのためには、**java.lang.reflect**パッケージの動的プロキシ機能を使用して、アプリケーション・コンポーネントのコンテキストを、作成するオブジェクト・インスタンスに関連付けます。このサービスを使用して、カスタムの並列処理アプリケーションおよびサービスを開発し、**ExecutorService**の実装などのカスタムJava EEコンポーネントを作成できます。

コンテキスト依存オブジェクトのプ

ロキシ・インスタンスを作成するためには、[ContextService](#)インスタンスの[createContextualProxy\(\)](#)メソッドを呼び出します。タスク・インスタンスは、[Callable](#)インタフェースを実装する必要があります。この[call\(\)](#)メソッドにタスクの実装を記述します。本記事の例では、スレッド・プールのマネージド・エグゼキュータを生成するために使用できるカスタムのSingleton EJBの生成方法を示します。**リスト9**は、スレッド・プールのマネージド・エグゼキュータを生成するために使用するSingleton EJBである[ExecutorAccessor](#)のコードです。このEJBを使用して、コンテキスト依存のタスク・インスタンスを渡します。

この例のタスク

は、[AcmeSingleReservation](#)というクラス内に実装します ([リスト10](#))。

`AcmeSingleReservation`は、予約ID番号を受け取り、データベースに問い合わせ、一致した予約レコード・オブジェクトを返します。このクラス内の[call\(\)](#)メソッドでは、データベース接続を作成し、指定された予約を問い合わせ、オブジェクトを返します。

タスクはステートレス・セッションEJBによって起動されます。このEJBが特定の予約ID番号を[AcmeSingleReservation](#)タスク・クラスに渡すと、予約オブジェクトが返されます。各予約を取得するために、複数のスレッドが使用されます。[ContextService](#)は、[AcmeSingleReservation](#)の呼出しごとにコンテキスト依存プロキシを作成するために使用します。そのプロキシは[ExecutorAccessor](#) Singleton EJBから取得されたスレッド・プールに対して送信

リスト7

リスト8


注:このリストは、紙面の都合上抜粋になっています。省略部分は...記号で示しています。コード・リストの全体は、本号のコード・リストをダウンロードしてご確認ください。

```
public class AcmeReservationCount implements Runnable,
Serializable {
    ParkReservationFacade parkReservationFacade =
lookupParkReservationFacadeBean();
    String reportName;
    ...

    public void run() {
        runCountReport();
    }

    /**
     * Prints a count of reservations.
     */
    public void runCountReport() {
        System.out.println("Park Reservation Count for Today");
        System.out.print
In("=====");
        Long reservationCount =
parkReservationFacade.findCount();
        System.out.println(reservationCount);
        // Email in production application
    }

    private ParkReservationFacade
lookupParkReservationFacadeBean() {
        try {
            Context c = new InitialContext();
            return (ParkReservationFacade) c.lookup(
"java:global/JavaMagazineEE7/ParkReservationFacade");
        } catch (NamingException ne) {
            ...
        }
    }
}
```

 [すべてのリストのテキストをダウンロード](#)

されます。このスレッド・プールの`take()`、`get()`メソッド・チェーンを呼び出すことで、予約オブジェクトが返されます。**リスト11**に、この実装を含むステートレス・セッションBeanである`ReservationChecker`の実装全体を示します。

サーバー・スレッド

サーバーサイドのスレッド・インスタンスは、`javax.enterprise.concurrent.ManagedThreadFactory` インタフェースを使用して作成できます。サーバーサイド・スレッドの生成は、アプリケーション内でボトルネックを発生させずに長時間実行タスクを実行する必要がある場合に便利です。新しいスレッドは、`java.util.concurrent.ThreadFactory` インタフェースの `newThread(Runnable r)` メソッドに、実行する `Runnable` クラスを渡して呼び出すことで作成できます。作成後、その新しく作成したスレッドの `start()` メソッドを呼び出して、アプリケーション・コンポーネント・インスタンスのコンテキスト内でスレッドを開始できます。

たとえば、Acme Worldアプリケーションで[RestaurantReservation](#)エンティティを定期的にポーリングして新しい予約があるかを確認し、新しい予約がある場合にアラートを渡すことにします。データベースのポーリングとアラートの作成を行うプロシージャを格納するための[Runnable](#)クラスを作成できます。**リスト 12**に、この例のマネージド・スレッドを作成するために使用する[Runnable](#)のコード([ReservationAlerter](#)という名前のクラス)を示します。[ManagedThreadFactory](#)の[newThread\(\)](#)メソッドを呼び出して、このクラスをスレッド内に設定できます (**リ**

スト13)。

リスト14に、@Resourceによるインジェクションを経由してManagedThreadFactoryへの参照を取得し、新しいThreadインスタンスを作成して開始するためのコード全体を示します。

トランザクションの利用

並列処理に関するリソースでは、[javax.transaction.UserTransaction](#)インタフェースを使用した、ユーザー管理のグローバル・トランザクション境界(デマーケーション)をサポートしています。このサポートによって、[begin\(\)](#)メソッドによるトランザクションの開始、[commit\(\)](#)メソッドによるコミット、[rollback\(\)](#)メソッドによるロールバックを必要に応じて実行できます。マネージド・タスクのインスタンスは、渡されたスレッドのトランザクション・スコープ外で実行されます。そのため、トランザクションを利用するアクションの調整をエグゼキュータが行う必要があります。また、実行中のスレッドでアクティブであるトランザクションはすべて一時停止状態になります。**リスト15**に、1つのトランザクション内においてタスクからトランザクション対応リソースを操作する方法を示します。

リスト9 リスト10 リスト11

```
@Singleton
public class ExecutorAccessor {

    private ExecutorAccessor executorAccessor;
    private ThreadPoolExecutor tpe;
    @Resource(name = "concurrent/__defaultManagedThreadFactory")
    ManagedThreadFactory threadFactory;

    @PostConstruct
    public void postConstruct() {
        tpe = new ThreadPoolExecutor(
            5, 10, 5, TimeUnit.SECONDS,
            new ArrayBlockingQueue<Runnable>(10),
            threadFactory);
    }

    public ExecutorService getThreadPool() {
        return tpe;
    }
}
```

 すべてのリストのテキストをダウンロード

まとめ

マルチスレッド対応型エンタープライズ・アプリケーションを取り扱うためにカスタム・ソリューションを開発する時代は終わりました。これからは、Concurrency Utilities for Java EEが、エンタープライズ・アプリケーションにおける並列処理の標準的なソリューションになります。このソリューションは、単純かつ高度な並列処理のパターンをサポートします。Java SEの並列処理APIの使用に精通する開発者は、Concurrency Utilities for Java EEの利用方法を問題なく習得できるでしょう。いずれのAPIも同様のガイドラインに従っているためです。また、初心者にとっても、このAPIは単純で直感的です。

本記事で使用した例を実際に確認するためには、[記事のソース](#)よりNetBeansプロジェクトをダウンロードし、作成します。その後、Apache Derbyを使用してデータベースを作成します。Apache DerbyはGlassFishにパッケージ化されています。最後に、お使いのGlassFishサーバーにこのアプリケーション・プロジェクトをデプロイしてください。

LEARN MORE

- [The Java EE 7 Tutorial](#)
- [JSR 236 : Concurrency Utilities for Java EE](#)

リスト12

リスト13

/リスト14

/リスト15

注：このリストは、紙面の都合上抜粋になっています。省略部分は…記号で示しています。コード・リストの全体は、本号のコード・リストをダウンロードしてご確認ください。

```
public class ReservationAlerter implements Runnable {
```

```
@Override
public void run() {
    while (!Thread.interrupted()) {
        reviewReservations();
        try {
            Thread.sleep(100000);
        } catch (InterruptedException ex) {
            // Log error
        }
    }
}
```

```
public Collection reviewReservations() {
    Connection conn = null;
    Properties connectionProps = new Properties();
    connectionProps.put("user", "user");
    connectionProps.put("password", "password");
    Collection reservations = null;
    try {
        // Obtain connection and retrieve reservations
        conn = DriverManager.getConnection(
            "jdbc:derby:acme;create=false",
            connectionProps);
        // Use the connection to query the database for reservations
    } catch (SQLException ex){
        System.out.println("Exception: " + ex);
    } finally {
        ...}
    return reservations;
}
```



すべてのリストのテキストをダウンロード



ERIC J. BRUNO

ソフトウェア・アーキテクチャや開発のトピックを扱うライター、編集者。IT コミュニティにおいて 20 年以上の実績がある。デスクトップからデータセンターに至るさまざまなトピックについて講演。世界中の企業でエンタープライズ・アーキテクト、開発者、業界アナリストも務める。

「Internet of Things」入門

実際に組み込みアプリケーションを作成することで Internet of Things に関連する概念や問題を理解する

多くの企業は、身の回りのあらゆるスマート・コンピューティング・デバイスが相互に通信し、さらによく利用するエンタープライズ・アプリケーションとも通信するような未来を約束しています。この手法は、マシン・ツー・マシン (M2M) アーキテクチャとも呼ばれます。オラクルも、Internet of Things (モノのインターネット：IoT) を重視した計画を立てています。このような IoT の約束を実現することは、モバイル・デバイスで Bluetooth や Wi-Fi をオンにする程度の単純なことに思えるかもしれませんが、しかし、すべてが接続された未来の世界を最大限に活かすためには、もっと深い部分で統合する必要があります。

デバイスとは

IoT のアーキテクチャや戦略の検討における最初の手順は、デバイスとは何かを定義することです。ほとんどの人はまず、デバイスをスマートフォンやタブレットと定義しますが、より小さく低機能の携帯電話もデバイスとして適格です。センサー、電気制御装置、

RFID リーダー、最新の家庭用電化製品なども、M2M や IoT の世界においてそれぞれの役割を果たします。基本的に、一意に識別でき、適切なリソースやロケーションに関するデータを持つすべての電子的なエンドポイントは、IoT アーキテクチャにおける有効なデバイスとして適格です。

デバイスの種類によってデータ量や実用性は異なりますが、大量のデータを組み合わせることで、非常に単純なデータからでも途方もない価値を引き出すことができます。重要なのは、デバイス同士の通信や、さらには協調動作までも可能になるように、すべてのデバイスを識別し、まとめて接続することです。この識別と接続こそ、インターネットが重要な役割を果たす分野です。

Internet of Things

1999 年、マサチューセッツ工科大学 Auto-ID センターの共同創設者である Kevin Ashton 氏は、Internet of Things という用語を考案し、あらゆる場所にあるセンサーを通じて、インターネットが実世界に接続されるシステムと定

義しました。Ashton 氏の構想は、データ入力を人間だけに頼らずに、独自のデータを収集するコンピュータ、デバイス、センサーによる地球規模のシステムを構築することでした。結果として、この地球規模のシステムでは、何があるのか、どの場所でどの時点で何が起きているのかを把握できます。その先につながるのは、大幅な無駄の削減、コストの低減、人間が機械を操作することによる損失（定義は人が行うのですが）の排除が可能となる、相互に接続されたシステムの世界でしょう。

研究者は、アドレス指定能力、通信規格、電池残量、データ収集などの問題を懸命に解決しようとしています。一方、各企業も、ギャップを埋め、標準プラットフォームを定義し、すべての関係者がメリットを享受できるエコシステムの構築を急いでいます。IoTはごく初期段階にあ

り、現在はIoTの成立に関わることのできる面白い時期です。

Java と IoT：ハイパーコネクティビティ向けのプラットフォーム

Java は組込みデバイス上で誕生しました。当初、Star7 携帯端末に搭載される Green Project の一部であった Java は、その後、Web と組込みの双方のドメインにおいて大きな成功を収めました。おそらく、IoT 戦略を推進する今日のテクノロジーの中では、Java がもっとも良い位置にあるでしょう。

CPU やメモリに制限のある

フィーチャーフォンから、処理能力や容量が豊富なラックマウント型サーバーに至るまで、Java はさまざまなデバイス上で稼働できます。そのことを考慮すれば、Java は、ユビキタスな接続性、つまりハイパーコネクティビティを実現したコンピューティング・リソースの世界を推進することを定め

成功に不可欠
おそらく、IoT 戦略を推進する今日のテクノロジーの中では、**Java がもっとも良い位置にある**でしょう。

データを統合し、データに対してマップ / リデュース操作を実行し、Hadoop 分散ファイル・システム（HDFS）などのさまざまなファイル・システムをサポートする必要があります。

前述のイベント処理に関する説明のとおり、データの収集や整理を行う場所と物理的に近い周辺部で分析を実行することで、応答時間の短縮が可能な、より階層的なシステムを構築できる場合があります。

IoT の具現例

組込み、IoT というシナリオにおける Java の例として、512MB の Raspberry Pi を使用して家庭用ゲートウェイ・サーバーの基礎を構築します。このゲートウェイは、家庭において、電気機器を遠隔操作したり、食器洗い機などの電化製品をプログラミングし、電気使用料の単価（ワット時あたり）が特定のしきい値よりも低い場合に限り電源を入れるなどの目的で使用されることが想定できます。このゲートウェイを、電力会社のゲートウェイであるスマート・メーターと統合することで、消費者が電力コストの低減によるメリットを得て、電力会社は電力需要をより効率的に管理できるという意味で、すべての人にとって有益な真のIoT ソリューションを構築できます。

より現実的なソリューションを構築するために、ブラウザからアクセスして制御できるように、アプリケーションをJava サブレットとして配置します。そうすることで、セキュリティ上の課題はあるものの、どこにいてもデスクトップ、ラップトップ、またはスマート・デバイスから家庭用ゲートウェイにアクセスで

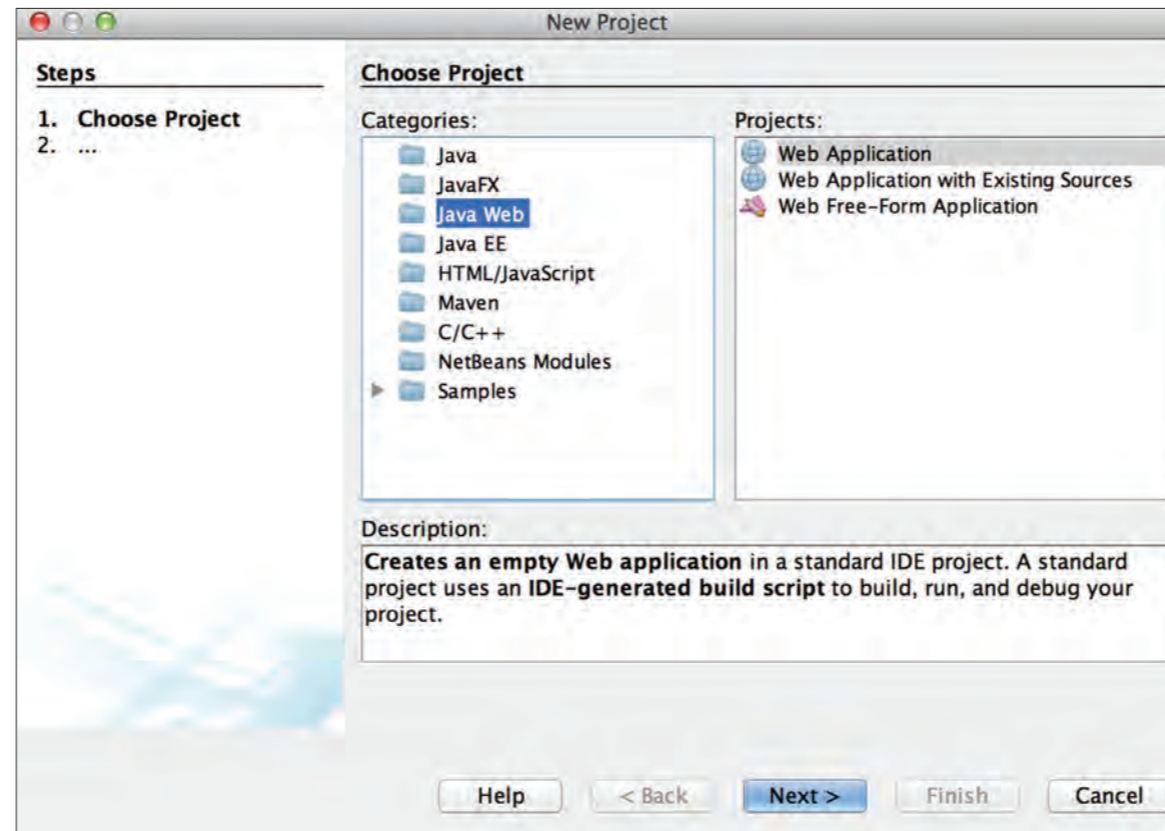


图 2

きます。

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。

少し一服：組込みの環境においては、Java ME と Java SE から選択できます。ターゲット・デバイスの機能やアプリケーション、その他の要因から判断します。たとえば、Java ME では RAM などの必要なリソースが少なくなります。しかし、本記事の例では、Oracle Java SE Embedded を選択します。わずか 32MB のメモリで実行できる Oracle Java SE Embedded は、標準的な Java の要件と組込み開発の要件との絶妙なバランスをとっています。さらに、このサンプルの要件である、Java サブレッツ

トを搭載した組み込み Web サーバーを実行するという要件を満たしています。

まずは、[Oracle Java SE Embedded](#)をダウンロードしてインストールします。次のコマンドを使用して、ダウンロードした .tar ファイルを Raspberry Pi のホーム・ディレクトリに解凍します。

~ \$ tar xvf <filename>.tar
~ /ejre1.7.0_40/bin/java -version を
実行してインストールをテストできます
(ディレクトリ名は実際のインストール・
ディレクトリの名前に置き換えてくださ
い)。

次に、サーブレット・アプリケーションを実行するためには、Apache Tomcat、IBM の WebSphere Liberty

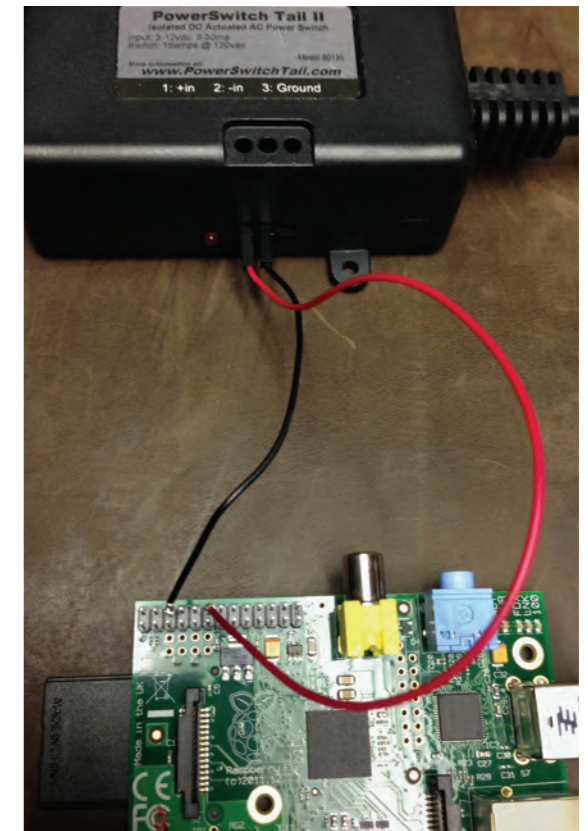


图 3

[Profile](#)、Oracle Java Embedded Suite などの、組込みの環境で実行するように設計されたサーバーを使用できます。サンプルのサーブレットのコードは標準の Servlet 仕様に準拠しているため、各サーバーで同様に動作します。ただし、配置方法のみがやや異なります。本記事では、コード自体について説明します。

まず、NetBeansで「Java Web」タイプの新規アプリケーションを作成します(図2)。

筆者が試した際には、GlassFishをターゲット・ホストとして選択し、Raspberry PiでサブレットをホストするサーバーとしてOracle Java Embedded Suiteを使用しました。この構成によって、ローカルで開発しRaspberry Piと同様の配



ヒント：型パラメータはコンパイル時に削除されます。

2 コード

Sample クラス内で修正すべき個所はどこですか。

```
import java.util.ArrayList;
import java.util.HashMap;

public class Sample{

    public <T> void setArrayList(ArrayList<T> arrayList, T t){
    }

    public <T> void setArrayList(ArrayList arrayList, Object obj){
    }

    public <T extends HashMap<K,V>> void setArrayList(ArrayList<T> arrayList){
    }

}
```

3 正しい修正方法はどれですか

- 1) `setArrayList` という名前のメソッドを 3 つ宣言することはできないので、最後の `setArrayList` メソッドを削除する。
- 2) 型パラメータ `T` は `HashMap<K,V>` を継承できない。
- 3) ジェネリクスของメソッドを含む場合はクラスもジェネリクスでなければならない。
- 4) 最初の 2 つのメソッドのうちのどちらかを削除し、3 番目のメソッドに型パラメータ `K` および `V` を追加する。

2013 年 9 月 / 10 月号で、多言語開発者の Attila Balazs がクラスの初期化に関するコードの課題を出しました。Balazs はデッドロックになるか NullPointerException をスローして、最後まで正常に実行されることのないコードを提示し、その修正方法を問いました。正解は 1 番と 3 番の両方です。問題の根本原因は、2 つの異なるスレッドからクラスローダーを呼び出せるため、デッドロックが発生することにあります。

- App スレッドから App を参照しているため、このクラスのロードと初期化が必要です。しかし、この初期化の処理中に Controller をロード（および初期化）する必要があります。
- このクラスはすでに main からロードしているため、main スレッドは App スレッドによる App クラスのロード / 初期化を待機しますが、同時に App スレッドは main による Controller クラスの初期化を待機します。

Integer を int に変更すると、コンパイラは RETRIES を定数とみなしてインライン展開するため、2 つのクラス間にある依存関係が削除されます。また、RETRIES を Controller 内に移動しても (RETRIES は Controller クラス内でのみ参照されているため、本来このクラス内にあるべき)、この依存関係が再び壊れます。

今回は、Oracle ACE であり NuBean のコンサルタントである Deepak Vohra 氏からの出題です。

1 問題

ジェネリクスが Java に追加されたのは比較的最近 (Java SE 5 から) であるため、その使用法は必ずしも明確ではありません。

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