

Java™ magazine

By and for the Java community 

海に浮かぶJAVA

JAMES GOSLING氏、Liquid Robotics「Wave Glider」で
新たな進路を描く

18
DUKE'S
CHOICE
AWARDS



34
BRIAN
GOETZ、PROJECT
LAMBDAを語る

//table of contents /

COMMUNITY

02

From the Editor

03

Java Nation

08

JCP Executiveシリーズ

Ben Evansに聞く

ロンドンJUG代表のBen Evans氏がJCPとJavaコミュニティについて語る。

JAVA TECH

23

New to Java

BlueJを使用したオブジェクトの相互作用に関する学習

オブジェクト指向プログラミングにおける抽象化とモジュール化

28

New to Java

Webサービスのセキュリティ入門サーバーからクライアントまで

SSLを使用したトランスポートセキュリティの設定によるWebサービスのセキュリティ向上

40

Java Architect

Java HotSpot VMの内部を探る(2):パフォーマンス解析のための統計情報

マクロレベルのベンチマーク、分布形状の理解、そしてクライアント視点による計測が重要な理由

44

Java Architect

NIO.2の新ファイル・システムAPIの紹介

Java SE 7でリリースされた新ファイル・システムAPIを活用するreleased in Java SE 7

50

Enterprise Java

Java EE Connector Architecture 1.6

コンテナ・サービスとの無駄がなく深い統合を実現する

58

Mobile and Embedded

Payment APIJSR 229入門

Java MEアプリケーション/ゲームでのJSR 229の利用法を学ぶ

64

Mobile and Embedded

Oracle Berkeley DB

Java EditionのJava API

Oracle Berkeley DBの基本APIの動作を学習する

75

Polyglot Programmer

GRAAの将来

Oracle LabsのDr. ThomasWuerthinger、Graalプロジェクトの目標について語る

78

Fix This

JavaFX Media APIコードの問題に挑戦



12

Java in Action

海に浮かぶJAVA

Liquid Robotics、JavaのパイオニアJamesGosling氏の力を得て新たな進路を描く

18

Java in Action

DUKE'S CHOICE AWARDS

初のCommunity Choice Award受賞者をはじめとした、本年の多様な受賞者を紹介

34

Java Architect

PROJECT LAMBDAの展望

Java SE 8で導入されるラムダ式の重要性についてJava言語アーキテクトのBrian Goetzに聞く

69

Rich Client

DATAFX

実世界データをJavaFXコントロールに展開

PHOTOGRAPH BY BOB ADLER

COMMUNITY

JAVA IN ACTION

JAVA TECH

ABOUT US

0

f

Twitter icon

Java.net icon

blog

Java logo

01



Caroline Kvitka 2001年オラクル入社
編集長としてOracle Magazine、Profit、Java Magazineの刊行に従事。
90年代初頭からテクノロジー関連の記者を務め、また、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。

イノベーションはIT業界の生命線である—Java Community Process Executive CommitteeのメンバーであるBen Evans氏はインタビューでこう話しました。そして多くのケースで、イノベーションが起きるところにJavaがあります。本号では、2012年Duke's Choice Awardsの受賞者を紹介することで、イノベーションにあふれるJavaの使い方を讃えたいと思います。10周年を迎えた同賞では、審査基準自体にも小さなイノベーションが起こりました。受賞者の多くはJavaを使用した技術的イノベーションを認められましたが、2組のJavaユーザー・グループ(JUG)については、Javaコミュニティの発展につながる革新的なアプローチが認められたのです。同賞で2組のJUGが選出されたことは、Javaが企業主導ではなくユーザー主導であり続けていることを示しています。[ぜひ受賞者たちの紹介記事をお読みください。](#)

イノベーターと言えば、本号の特集記事では同じくDuke's Choice Awardを受賞したLiquid Roboticsについて取り上げています。同社では、JavaのパイオニアであるJames Gosling氏がチーフ・ソフトウェア・アーキテクトを務め、Wave Gliderという海洋データ収集に使用する海洋ロボットの内蔵ソフトウェアを(Javaで)書き直しています。Gosling氏は長年の友人であるLiquid RoboticsのBill Vass氏の元を訪ねたときに、Wave Gliderのテクノロジーは非常にクールだと思い、自分もここで働けないかと尋ねたそうです。詳しくは「[海に浮かぶJava](#)」を参照してください。

Java SE 8は現在検討中であり新しいイノベーションに満ちています。その中の1つがラムダ式です。Java言語アーキテクトのBrian Goetz氏へのインタビュー記事「[Project Lambdaの展望](#)」で、ラムダ式によってコードの書き方がどう変わるかについて学びましょう。

イノベーションはまた、変化をもたらします。初期のJava Magazine編集長であるJustin Kestelynは、別のコミュニティ育成の機会があつて異動しましたが、私たち編集部はJava Magazineの発行を重ねるごとに、その内容を充実させていきたいと考えています。私はJustinから編集長の職を受け継いだことを光栄に思い、また、これまでの彼の数え切れない貢献に感謝します。

多くの読者の皆さんとJavaOneでお会いできるのを楽しみにしています(JavaOneでは必ずイノベーションと出会えますよ)。

編集長Caroline Kvitka 

写真:BOB ADLER

//フィードバックをお送りください!

今後の改善のために、いただきました提案はすべて検討いたします。提案の数が多い場合、直接の返信をいたしかねることがありますのでご了承ください。



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®





Greenfootを使用したプログラミングについて説明するケント大学のMichael Kölling氏(左) 未来の開発者たちがプログラミングを経験している



JAVAで創る未来

サンフランシスコのベイ・エリアで300名以上の学生が、8月7日～8日の2日間に開催されたインタラクティブ・ワークショップ「Make the Future Java (未来のJavaを創る)」に参加しました。このワークショップは、テクノロジーやコンピュータ・サイエンスに対する学生の情熱を刺激し、次世代のJava開発者を生むことを目的に開発されたものです。カーネギーメロン大学およびケント大学の教授が、Alice (アニメーション作成のための教育用ビジュアル3Dソフトウェア) とGreenfoot (ゲーム/シミュレーション作成のための教育用ビジュアル2Dソフトウェア) を実際に使用するセッションを開きました。Oracle Education Foundationは先日、カーネギーメロン大学のAliceプログラムおよびケント大学のGreenfootの取り組みに対して助成金を交

写真:SAUL LEWISLEWIS

付しました。学生の関心を引き起こし、未来の技術者やイノベーターの中に眠るコンピュータ・サイエンス、テクノロジー、Javaへの情熱を育む支援を行うためです。Oracle Academyバイス・プレジデントのAlison Debenwick-Millerは「オラクルはJavaの幹事として、Javaテクノロジーに対して投資し、また、JavaやJavaがもたらすチャンスについて次世代の人々を教育する創造的な手法に対して投資しています。次世代のJava技術者やイノベーターとなる人たちがどんどん現れています。」と語ります。[Make the Future Javaの詳細はこちらから。](#)

JAX Innovation Awards 2012



7月10日、カリフォルニア州サンフランシスコのJAX Conferenceで、2012年のJAX Innovation Award受賞者が発表されました。同賞は、Javaエコシステムにおけるイノベーションの文化を讃えるものです。受賞者は3段階のプロセスを経て選出されます。まず、開発者コミュニティが受賞候補者をノミネートし、次に独立したエキスパート・パネルが各受賞カテゴリーの最終選考者を選出します。最後に開発者コミュニティが投票を行います。

2012年の受賞者には、2,500米ドルの賞金が贈られました。本年の受賞者は、[Restructure101](#) [「Most Innovative Java Technology」部門]、[JetBrains](#) (「Most Innovative Java Company」部門)、[Adam Bien](#) 氏 (Top Java Ambassador 部門、写真左)、[Charles Nutter](#) 氏 (「Special Jury Award」部門、写真右) です。

写真:PIOTR MALECKI / GETTY IMAGES, MARKUS EISELE



TDC: THE DEVELOPER'S CONFERENCE



7月にブラジル・サンパウロで開催された **The Developer's Conference (TDC)** には、**3,400人**を超える開発者が参加しました。TDCはGlobalcodeが企画した5日間のイベントであり、ブラジルではマルチコミュニティ開発者カンファレンスの筆頭という位置付けです。企画者の一人、**Yara Senger**氏は次のように話します。「私たちは『マルチテクノロジー』ではなく、『マルチコミュニティ』という言葉が好きで使っています。さまざまなコミュニティが一緒になると面白いし、有益な時間を過ごせるからです。それぞれのコミュニティに個性があり、お互いのコミュニティから多くのことを学んでいます」

TDCでは37のトラックで250人以上の講演者が発表を行いました。今年のTDCでの新たな試みは、オラクルが支援する **Java University** トラックです。このトラックでは、Javaを知らない大学生やプロフェッショナルを対象として、教育に主眼を置いた入門レベルのレクチャーや実践エクササイズが行われました。ブラジルのさまざまなJavaユーザー・グループ (GoJava、JavaBahia、JavaNoroeste、SouJavaなど) が、Java、SOA、Python、Ruby、モバイル、デジタルTVなどを扱うトラックでそれぞれの専門性を発揮し、TDCを教育的なイベントとして成功に導きました。モバイルに関するトラックでは、Java MEのアプリを開発した開発者グループ (15歳の学生を含む) にNokiaの携帯電話が贈られました。

TDCの基調講演では、Oracle University ブラジル統括マネージャーである **Debora Palermo** が、オラクルの **Workforce Development Program (WDP)** について説明しました。「WDPでは各教育機関がオラクルのトレーニングを実施できます。WDPによって、世界中の地域コミュニティでオラクルのトレーニングを容易に、かつ低コストで受けることができます。Oracle Universityは次世代のJava開発者の育成に力を入れており、それを実現できるのがWDPです。」Oracle UniversityはGlobalcodeとの協力の上でWDPを展開しています。学生はWDPを通じて、オラクル公式のコース認定を取得できます。

写真提供:GLOBALCODE



TDCトラックの主催者とカンファレンスへの協力者が最終日にステージに上る(上) トレーニング・プログラムについて説明するOracle UniversityのDebora Palermo(下)

オープン・データ、 政府、透明性

オープン・データの動きが世界中で拡大しています。オープン・データ構想の1つであるOpen Government Partnershipは現在、世界中にある政府のデータを市民が検索/アクセスしやすくなるように取り組んでいます。ブラジル・サンパウロのThe Developer's Conference(TDC)では「Open Data(オープン・データ)」という同カンファレンス初のトラックで、オープン・データを使用したアプリケーション構築のために開発者が知っておくべきことを重点的に取り上げました。このトラックの各リーダーが、成功を収めているオープン・データ・プロジェクトについて紹介し、**セマンティックWeb**、データ視覚化、API設計について討論しました。

トラックの最終日は「Transparency Hacker Day」として、既存の政府オープン・データを使用したアプリケーションの作成に充てられました。このオープン・データには、ブラジル政府がリリースしたほとんどの公務員の給与情報が含まれます。参加者は「Serve o Quanto Ganha(自分の給与の価値)」というアプリケーションを作成しました。このアプリケーションは、2人の公務員の情報を取得し、何人分のXの給与がYの給与を支払うために必要となるかを比較するものです。

Learn more about open data at the [米国政府、ブラジル政府、イギリス政府の各オープン・データ・プロジェクトと、2012年のInternational Open Government Data Conference](#)について詳細をご覧ください。



Rich Hickey



Rémi Forax



John Rose



Georges Saab

2012 JVM LANGUAGE SUMMIT

カリフォルニア州サンタクララのオラクル社屋内で7月30日～8月1日に開催された [2012 JVM Language Summit](#) では、世界を代表するJava仮想マシン(JVM)言語アーキテクトが一堂に会しました。「このビルが崩壊したら、言語のイノベーションが100年遅れてしまう」と、ある参加者がツイートするほどです。会場ではコードを入力するカタカタ音が鳴り響きました。

講演者は、**Georges Saab** 氏(イントロダクション)、**William Cook** 氏(バッチ処理)、**Mark Roos** 氏(RTalk)、**Matt Fowles** 氏(段階的コンパイル)、**Jochen Theodorou** 氏(Groovy、invokeDynamic)、オラクルの**John Rose** (配列)と**Brian Goetz** (Project Lambda)、**Dan Heidenga** 氏(MHイントロダクション)、**Lukas Stadlar** 氏(Truffle)、**Per Bothner** 氏(Kawa)、**Jeroen Frieters** 氏(CLR/JVM)、**Duncan MacGregor** 氏(JVM、ASMへの移行)、

Rich Hickey 氏(Datomic)、**David Chase** 氏(Fortress)、**Rémi Forax** (JDart)、**Basil Hosmer** 氏(Mesh)、**Ryan Sciampacone** 氏(マルチテナントJVM)、**Jim Laskey** 氏(JDI)、**Doug Simon** 氏(Graal)、**Michael Wiedeking** 氏(AL1 JVM アセンブラ)、**Andrey Breslav** 氏(Kotlin/Java 相互運用性)、**Gilles Duboscq** 氏(Graal)、**Donald Raab** 氏(コレクション) などです。

参加者の一人は「これまで参加した中で一番良いカンファレンスだった。宣伝や販売はなく、オープンな学習と好奇心という精神で、全員がただテクノロジーに集中していた。ここにいるのは、プログラミング言語の発案者たちだ」と話していました。また、別の参加者のTシャツには次のようなロゴがありました。「Changing lives one line at a time (1行ずつで変わる人生)」

プレゼンテーションのビデオが[オンライン](#)で公開されています。

JCP: オープン性と透明性

JCP.Next 関連のJSRでは、Java Community Process (JCP) に対して新たにオープン性と透明性の実現を要求しています。たとえば、[JSR 348](#)では、世界中の人々を1年に2回のJCP Executive Committee (EC) 会議に招待しなければならないとされています。1回目の会議は6月26日に開催されました。この会議の詳細は[こちら](#)でご覧になれます。

JCPはJavaOneでもセッションを持つ予定です。4つのセッション「JCP.Next: Reinvigorating Java Standards (JCP.Next: Java標準の再活性化)」、「101 Ways to Improve Java: Why Developer Participation Matters (Java発展の基本手法:なぜ開発者の参加が重要なのか)」、「Meet the JCP Executive Committee Candidates (JCP Executive Committeeの候補者紹介)」、「The JCP and OpenJDK: Using the JUGs' Adopt Programs in Your Group (JCPとOpenJDK:JUGのAdoptプログラムを自分のグループで利用する)」を主催します。また、JCPは3つのイベントを開催します。それぞれ、「JCP EC Face to Face Meeting (JCP EC対面会議)」、「JSR Expert Group Spec Lead Gathering (JSR専門家グループ仕様リード会議)」、そして10月2日のJCPパーティです。JCPパーティでは、今回で10周年となる[JCP各賞の年間受賞者](#)が発表されます。本年の受賞カテゴリは、「JCP Member/Participant of the Year (最優秀JCPメンバー/参加者)」、「Outstanding Spec Lead (最優秀仕様リード)」、「Most Innovative JSR (革新性に最も優れたJSR)」です。

注目のJAVA.NETプロジェクト

VISUALLANGLAB

VisualLangLab はコード/スクリーンショットの記述が不要な世界初のパーサ/ジェネレータ用の完全ビジュアル統合開発環境 (IDE) です。この IDE は

文法ルールを、ノードを含むツリーとして直観的なアイコンで表します。作成した「ルール・ツリー」は、ボタンをクリックしてすぐに実行でき、コード生成やそのほかの中間ステップは必要ありません。VisualLangLab を利用すればルールをビジュアルに表現できるため、特殊なスキルを持たないユーザーでも洗練されたパーサを作成できます。作成されたパーサは、任意の Java 仮想マシン言語で記述されたクライアント・プログラムに組み込むことができます。

VisualLangLab については、[Java Posse Episode 364](#) で次のように強調されています。「今続々と登場している新言語の成果を見て『私の方がうまくできる』と思う人は、今こそチャンスです。VisualLangLab は言語の文法を設計するためのグラフィカル・ツールです。成熟した言語だけを対象にしたツールではありません。自由形式のドメイン固有言語を開発するためにも、このプロジェクトを利用できます」

VisualLangLab プロジェクトは 2011 年 1 月に開始しましたが、そのルーツは 2004 年にまで遡りま

す。この頃、同プロジェクト創設者の **Sanjay Dasgupta** 氏は、カスタムのビジネス指向言語向けに一連のツールを開発するチームを率いていました。作業が進むにつれ、Dasgupta 氏は従来のパーサ/ジェネレータ (ANTLR、JavaCC) は多くのユーザーにとって複雑すぎると感じました。そのため、Dasgupta 氏は VisualLangLab の前身にあたるツールの開発を始めました。

これまでに VisualLangLab をダウンロードした開発者は 1,400 人以上います。数年にわたる開発期間の中で、同プロジェクトの開発に携わったのは Dasgupta 氏だけです。おそらくコードが複雑なために、他の開発者を遠ざけているのではないかと Dasgupta 氏は語ります。しかし、彼は現在、コードの内部仕様ドキュメントの情報を補うことで、プロジェクトへの参加を希望する人が参加しやすい状態を作ろうとしています。

詳細は、「[VisualLangLab—Grammar Without Tears](#)」をご覧ください。また、[チュートリアル](#)、[FAQ](#) などの同プロジェクトの情報については [Java.net](#) にアクセスしてください。



JAVA CHAMPIONのご紹介

KIRK PEPPERDINE

Java Magazine: 育った場所はどこですか？

Pepperdine: カナダ・オンタリオ州のオタワです。オンタリオ州クリントンの学校に通い、その後ニューブランズウィック州のセントジョンに移って、またオタワに戻ってきました。

Java Magazine: 最初はどのような経緯で、プログラミングに興味を持ったのでしょうか？

Pepperdine: 生化学研究の 1 年目を終えた頃に景気が悪化し、夏休みの仕事がなかったので、プログラミング講習を受けました。それでピンときて、自分の生化学研究に役立てようとコンピュータを使い始めました。

Java Magazine: あなたが最初に習得したプログラミング言語は何ですか？

Pepperdine: 最初に習得した言語は PL/C で、IBM 370 で動かしました。その後すぐに大学での研究のために Fortran に乗り換えました。業務で最初に使った言語は BASIC でした。



Java Magazine: は、最初のプログラミングの仕事はどのようなものでしたか？

Pepperdine: 生化学者として働き始めた頃、大量のプログラミングが必要でした。思い出深い実験が 1 つあって、高周波振動換気回路の圧力リリース弁を制御するソフトウェアを開発していました。このプログラムを、Tiny BASIC を実行するチップ上の ROM に書き込む必要がありました。入力情報は、気圧からデジタル信号を作成する別の Intel チップから取得しました。この 2 基のチップと 1 つのソレノイドを 1 枚のボード上にワイヤラップして、次に人工気道の電気回路に組み込みました。人工気道は

暖かく、湿度が非常に高く、気体には塩分が含まれています。このボードは約 4 時間動作するのですが、その後は洗って乾かして、加圧滅菌する必要がありました。すごく楽しかったですね。

Java Magazine: 今後の数年間はどのようなものにしたいですか？

Pepperdine: アプリケーションのチューニング方法に関する開発者の考え方を変えるために努力を続けたいと思います。私は最近、パフォーマンス・ツールを提供する [jClarity](#) という会社を立ち上げました。jClarity では、他の会社では見られないようないくつかの実験的なソフトウェアを開発しています。

[Kirk Pepperdine](#) 氏についてはこちらで詳しく取り上げています。また、同氏の Java パフォーマンス・チューニング講座については [Kodewerk](#) をご覧ください。

LAMBDA EXPRESSION:

プログラミングのお供に音楽を



前号のJava MagazineでJavaをテーマにした歌を募集したところ、**Freddy Guime**氏より、ワンマン・バンド「**Lambda Expression**」について、現時点の曲一覧

とともにご連絡いただきました。Guime氏の最新の(かつお気に入りの)歌は、エッジな「**In the Zone**」です。この曲で彼は、すてきなプログラミングの日々について歌っています。何もかもが流れるように進み、コードがほとんど勝手にできあがっていくような感覚になり、ミーティングやメール、生活によって中断されたくないような日々です。また、Guime氏はJava Pub Houseでブログやポッドキャストを投稿しています。Javaをテーマにした歌がある方は、ぜひご連絡を!

OSCONが提供する

ハード・スキルとソフト・スキル

OSCON (オープンソースのカンファレンス) が7月にオレゴン州ポートランドで開催されました。3,600人以上が参加し、さらにライブ・ストリームを閲覧したバーチャル参加者は4,600人を超えました。OSCONでは、オープンソースのテクノロジーやコミュニティ、戦略に関する有益な情報が提供されました。「**High Performance Network Programming on the JVM** (JVMでの高パフォーマンス・ネットワーク・プログラミング)」、「**Hacking JavaFX with Groovy, Clojure, Scala, and Visage** (Groovy, Clojure, Scala, Visageを使用したJavaFXハック)」、「**The Art of Metaprogramming in Java** (Javaでのメタプログラミング術)」、「**The Java EE 7 Platform: Developing for the Cloud** (Java EE 7プラットフォーム:クラウドのための開発)」などのセッションでJavaに関する豊富な情報が発表されました。OSCONでは「ソフト・スキル」も重視しており、コミュニティやギークのライフスタイルに関するトラックもありました。「**Open Source 2.0: The Science of Community Management** (オープンソース2.0:コミュニティ管理の科学)」や「**Analyzing How Developers Learn Online for Fun and Profit** (趣味と実益のための開発者のオンライン学習方法分析)」などのセッションが執り行われました。これらのセッションは、各コーディングでオープンソース・コミュニティに貢献することはできるものの、長期的には他者との交流が成功に大きく影響するということを開発者に気付かせるような内容でした。発表のスライドとビデオは oscon.com で閲覧できます。



オープンソース・コミュニティ管理の科学について語る David Eaves氏



優れた JAVA ブログ 5選

世の中には、Java や Java 仮想マシン (JVM) テクノロジーに関する素晴らしいブログがたくさんありますが、最新情報や将来の予定について常に知るには、まずは次の5つのブログを確認すると良いでしょう。

- 1 **Adam Bien's Weblog**: 数々の執筆経験があり、Java Champion でコンサルタントでもある **Adam Bien** 氏が、自身のブログでJavaに関するほぼすべての情報を紹介し、特にJava EEについて重点的に取り上げています。
- 2 **FX Experience (JavaFX News, Demos and Insight)**: **Jonathan Giles** 氏がJavaFXに関するニュース、インタビュー、今週のリンク、リリース発表、プログラミングのヒントを紹介しています。
- 3 **Geertjan's Blog (Random NetBeans Stuff)**: **Geertjan Wielenga** のプリンシパル製品マネージャーであるGeertjan Wielengaのブログ。「NetBeansに関するブログであり、NetBeansについて述べることもあり、NetBeansに関連する話題についても掘り下げています。さらに、NetBeans自体についても...」
- 4 **Inspired by Actual Events (Dustin's Software Development Cogitations and Speculations)**: 開発者のDustin Marx氏が、Java、JVM、JavaFX、Groovyなどのさまざまな話題について見解を述べています。
- 5 **There's not a moment to lose!**: オラクルのJavaプラットフォーム・チーフ・アーキテクトであるMark Reinholdのブログ。あまり更新頻度は高くないですが、投稿された情報は大きなニュースであることが多いです。

画像:I-HUA CHEN





JCP Executiveシリーズ Ben Evansに聞く

ロンドンJUG代表のBen Evans氏がJCPとJavaコミュニティについて語る。Janice J. Heiss

写真: JOHN BLYTHE

この記事は、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第2回です。今回は、英国ロンドンのJavaユーザー・グループ(JUG)の代表を務めるBen Evans氏にお話をうかがいます。Evans氏は1990年代後半よりプロフェッショナルな開発に従事し、オープンソースの熱狂的な支持者でもあります。Evans氏はGoogleの初期株式公開(オークション方式としては史上最大)に向けたパフォーマンス・テストのリード・エンジニアを務め、また、BTと共に英国での3Gネットワークの初期実験に携わりました。ハリウッドの1990年代上位ヒット作のWebサイトをいくつか構築し、賞を獲得しています。英国で弱い立場にある人々を支援するためのテクノロジーのアーキテクチャ再設計や再検討を行った経験もあります。英国史上初のeコマース・サイトから数十億ドル規模の外国為替取引システムまで、あらゆるシステムに関わってきました。現在は、Javaパフォーマンス・チューニングの自動化を中心に扱うロンドンのスタートアップ企業iClarityのCEOを務めています。

Evans氏はMartijn Verburg氏とともに、Java SE 7について説明した『The Well-Founded Java Developer』を執筆しました。現在はロンドンのJUG「London Java Community (LJC)」の代表として、JCPの Executive Committeeに参加しています。



ベルギー・アントウェルペンのDevoxx 2011 Java開発者カンファレンスで、オラクルのBrian Goetz(左)、Mark Reinholdと共にステージ上に姿を見せるBen Evans氏

LJC、Duke's Choice Awardを受賞



London Java Community (LJC) は 2012 Duke's Choice Award を受賞しました。「Java テクノロジーの世界において極めて重大なイノベーション」を讃える同賞は、Java プラットフォームを使用する、もっともイノベーションに溢れた各種プロジェクトに贈られます。LJC は、Java エコシステムの発展につながる取り組みが評価されました。具体的には、JUG コミュニティを大規模で活気のあるものに成長させ、JCP Executive Committee のメンバーに選ばれた初の JUG となり、さらに Adopt-a-JSR プログラムや Adopt OpenJDK プログラムの陣頭指揮をとっている点が認められました。

Java Magazine: Java、ロンドンJUG、JCPとの関わりについて教えてください。

Evans氏: 私は1998年以来、Javaでプログラミングを行っており、現在はJavaパフォーマンス・チューニングの自動化を中心に扱うロンドンのスタートアップ企業jClarityのCEOを務めています。また、ロンドンJUGの運営を支援しています。ロンドンJUGは昨年、JCPのExecutive Committeeのメンバーに選出され、私自身もJCPには約18か月関わってきました。

私個人はロンドンJUGへの参加を通じてJCPに関わることになりました。Martijn Verburg氏とはロンドンJUGで出会いました。共に本を書き始めたところ、私は公の場で話すことが多くなり、コミュニティにも深く関わるようになりました。このようなきっかけで、JCPへの関わりが増えたのです。私は個人的に、LJCがコミュニティでの存在感を高めることを望んでいました。日常業務の中で、Javaは私にとって、複数利用している中の一言語に

過ぎませんでした。いつしか一番重要な言語になりました。そのため、積極的に関わって、Javaプラットフォームの潜在力を解き放ちたいと思いました。

LJCが最初にExecutive Committeeの選挙に立候補したときには、冷静に見て、「選ばれるわけがない。何人かの既存メンバーと意見が対立しているのだから」と考えていました。単純に世の中を少し騒がせるだけかもしれないが、もしかしたら何らかの議論を巻き起こせるのでは、とは思いました。ただ、正直、選挙に勝てるとは思っていませんでした。そのため、実際に選挙に勝ったと聞いたときは本当に驚きました。嬉しいハプニングでしたね。

Java Magazine: それぞれのJUGとその上位のJavaコミュニティとの関係に問題が発生した場合にはどう対処しますか。

Evans氏: Javaに関しては、いわゆる「氷山の一角」という問題があります。比較的積極性のあるJUGでも、JUGに参加していないの方が、JUGの参加者の数を大きく上回っています。ロンドンには約2,500人のJUGメンバーがいますが、私たちの見積りによれば、この数はロンドンにいるJava開発者の3~5%にあたり、非常に小さな割合です。

水面下には広大なJava開発者の氷山が広がっています。どう対処すべきかについてはまったくわかりませんが、これが現実です。2011年夏から秋にかけての調査でわかったこととしては、多方面で活躍する多くの人々が、特定のJSRやJavaの最新テクノロジーに関心を持っています。

Java Magazine: ロンドンJUGのミーティ





JCPのBen Evans氏は、Javaが「氷山の一角の問題」に陥っていると確信している。ロンドンのJava開発者のうちロンドンのJUGに参加している人の割合が少ないことがそれを裏付けている。

混乱について

単純に世の中を少し騒がせるだけかもしれないが、何らかの議論を巻き起こせるのでは、とは思いました。ただ、正直、選挙に勝てるとは思っていませんでした。

ングにはどのような人が参加していますか。

Evans氏: 本当にさまざまで、金融サービス会社や保険会社、あるいは電気通信分野のビジネスを手がける人も多くいれば、研究者も多く参加しており、大学生の数も増えています。最近では、ロンドン東部を拠点とするスタートアップ・コミュニティからの参加も見られます。

Java Magazine: JCPに対する参加を促進するにはどうすればいいでしょうか。

Evans氏: JCPが存在すること、JCPでの決定事項や新しい標準がJava開発者ひとりひとりの将来のキャリアに影響すること、さらにプロセスや標準策

定でJava開発者の助けが必要だということ、JCP側が常に発信する必要があります。参加の仕方は無数にあり、個人でも、企業でも、ユーザー・グループでも関わることができます。さらに、メンバーとして加わる方法もたくさんあります。たとえば、特に関心のある標準について進捗状況を追う、あるいは、特許の従来技術に関係する既存の標準や既存のOSS[オープンソース・ソフトウェア]プロジェクトについてJSR内で説明されていることを確認できます。また、新しい標準についてブログに書いて、人々の意識を高めることもできます。活動範囲を広げ、コミュニケーションを活性化させ、日々開発を手がける人にもっと多く関わってもらうこと。これらすべてが、より優れた標準の策定につながるでしょう。

JCPについてまったく聞いたことのないJava開発者、あるいはオラクルの買収後にJCPがなくなったと思い込んでいるJava開発者があまりにも多いのです。

Java Magazine: 標準の尊重とイノベー

ションの推進というバランスを保とうとする中で、JCPは一方向に大きく偏っていませんか。

Evans氏: この綱渡りは本当に難しいですね。イノベーションはIT業界の生命線ですが、互換性も強く求める必要があります。互換性がなければ、Javaプラットフォームの開発者やエンド・ユーザーがコストや外部性を押しつけられます。コストや外部性が大きすぎる場合は、みなJavaから離れていきます。私は、JCPは大部分正しい方向に向かっているけれども、このコストや外部性という要因を監視し続ける必要があると思います。

Java Magazine: 最近のJCPで一番良い出来事は何でしたか。

Evans氏: 昨年より、JCPへの関心が再度高まっています。多くの新しい血が注がれ、特に透明性を主要原則とするプロセスの新しいバージョンが登場し、さらに変化を求める勢いが新たに生まれています。この中で1つだけを選ぶのは難しいですね

Java Magazine: JCPにおいてどのような構造上の変化を求めていますか。

Evans氏: 2つの委員会を統合する動きや、JSPA[Java Specification Participation Agreement]でもっとも厄介な部分、はっきり言えば知的財産への対処やJCPでの知的財産の取り扱い方に関する部分を再検討する動きは、すばらしいことだと思います。これらは、変化が大幅に遅れている最重要課題です。

実務上の変化という観点では、JCPにはExecutive Committeeのメンバーに投票の日付について通知するための適切なカレンダー・システムがありません。この問題は必ず解決する必要があります。

Java Magazine: JCPの専門家グループのプロセスには満足していますか。どのような変化が必要だと思いますか。

Java Community Process でのオープン性と透明性の高まり

Java Community Process (JCP) では、3つのJSR (うち2つは検討中) によってオープン性と透明性が高まっています。

2011年10月に検討が完了したJSR 348 (別名JCP.next.1) では、透明性を高め、より広い範囲

からの参加を実現する、小さく単純だけれども重要な変更点について検討されました。「すでに新しいJSRや既存のJSRでJSR 348が課す新たな要件を導入しており、JSR 348の変更点の効果が始まっています」と語るのは、JCP会長のPatrick Curran氏です。

JSR 355 [JCP Executive Committee Merge (JCP Executive Committeeの統合)] は、効率性を高め、Java MEプラットフォームとJava SEプラットフォーム間の相乗効果を促すために、2つの執行委員会を1つに統合するものです。JSR 355の検討は2012年後半に完了する予定です。

JSR 358 (別名JCP.next.3) は2012年6月に提出されたJSRであり、JCPの大きな改定事項を制定するものです。このJSRではJava Specification Participation Agreement (JSPA) を改訂します。JSPAは、メンバーがJCP組織やJCPプロセス文書への参加時に署名する法的契約書です。多数の複雑な問題に対処する予定であり、多くの内容はJSR 348から持ち越されています。JSR 358は複雑であるため、完了予定は2013年末ですが、それ以後になる可能性もあります。

JSR 358で対処するトピックには、独立した実装 (リファレンス実装から派生していない実装) の役割、新しい透明性要件を適切に実施するためのライセンスとオープンソース、互換性の方針やTechnology Compatibility Kits (TCK)、個々のメンバーの役割、特許の方針、知的財産フローなどがあります。

すべてのJCP業務はオープンに実施されており、java.net で状況を確認し、参加することもできます。

Evans氏: 重要なのは、仕様リードの自主性と、標準プロセスを求める動きとのバランスを保つことです。個人的な考えとしては、リファレンス実装のライセンスやTCK [Technology Compatibility Kit] などの標準化された手順を用意すれば何らかの役に立つでしょう。また、仕様リードには標準化された手順の使用を義務づけるのではなく、使用を奨励すべきだと思います。

JCPはコミュニティと協力すべきです。コミュニティは材料やフィードバックを提供してくれます。そうすれば、コミュニティで何が流行しているか、どこが標準を必要としているかがわかります。この良い例がJSON標準です。よい成長を遂げていると受け止められている他の流行の言語と競争するためには、品質の良いJSONライブラリが必要です。

LJCは、コミュニティとの協力を推進するために、Adopt-a-JSRプログラムとAdopt OpenJDKプログラムを立ち上げました。この両プログラムでは、普通の開発者が、新しい標準や新しい言語テクノロジーの検討グループと直接やり取りできます。新しい標準や新機能を今後数年の間に日常業務で使用することになるエンド・ユーザーから、それらの標準や機能について早めにフィードバックを得ることができます。

Java言語を手当たり次第に変更することはできません。数百万もの開発者や、業界、企業に関わる世界中の多くの人々に影響を及ぼすからです。そのため、私たちJCPのメンバーは標準の番人として、ゆっくり慎重に進む必要があります。Scala言語のように、インターネットの速さで変更を加えることは不可能です。一方、エコシステム内の新しい言語は、ある種の言語研究所という役割を果たします。何が機能し、何が機能しないのかを、新しい言語から学ぶことができます。Java

のようにゆっくり進むことを強いられる状況にも良い点があります。

Java Magazine: 最後に、JCPの透明性やオープン性を高めるという約束をオラクルは果たしてきたでしょうか。

Evans氏: これまでオラクルが示してきたことはすべて前向きな内容でした。不満は特にありません。 </article>

Janice J. Heiss: オラクルのJava編集者であり、Java Magazineのテクノロジー編集者

LEARN MORE

- [Java Community Process](#)
- [Ben EvansのLondon Java Communityのブログ](#)

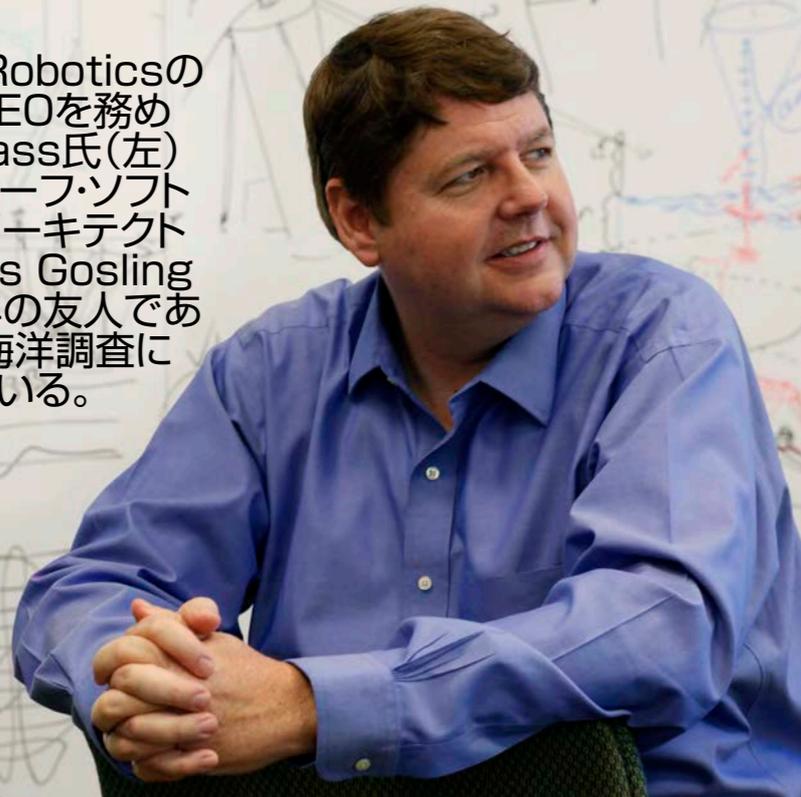
海に浮かぶJAVA

Liquid Robotics、Java
のパイオニアJames
Gosling氏の力を得て新た
な進路を描く
DAVID BAUM

カリフォルニア州モスラン
ディングで就航にのぞむ
Wave Glider

写真提供:LIQUID ROBOTICSUID ROBOTICS

Liquid Roboticsの社長兼CEOを務めるBill Vass氏(左)と同社チーフ・ソフトウェア・アーキテクトのJames Gosling氏は長年の友人であり、共に海洋調査に携わっている。



企業概要

LIQUID ROBOTICS, INC.

liquidr.com

本社所在地:

カリフォルニア州サニーベール

業界:

海洋データの収集、観察、観測、調査

従業員数:

90

使用しているJavaテクノロジー:

JDK 7
(NetBeans, Swing, NASA World Wind Server)

写真:BOB ADLER

Javaが成功している理由の1つが、その類いまれなる万能性です。インターネット・プログラムのアニメーションから携帯電話の基盤に至るまで、世界中でもっとも広く普及している言語です。中でも特に興味深いJavaプログラムが、科学調査用機器の「Wave Glider」に組み込まれています。Wave Gliderは、カリフォルニア州サニーベールに本社を置くLiquid Roboticsの主力製品であり、本年のDuke's Choice Awardを受賞しました(23ページの「Duke's Choice Awards」をご覧ください)。

Wave Gliderは、海洋波のエネルギーを推進力とする自走/自律型海洋ロボットです。水上の姿はあたかもハイテク機能を搭載したサーフボードのようですが、その実体は太陽電池とさまざまな種類の検査/測定機器で覆われた浮遊式プラットフォームです。このプラットフォームの下には、波の上下動を推進力に変えるための翼状パネルを複数備えた水中グライダーが潜んでいます。特許を取得したこの海洋波動力システムは、バッテリーや化

石燃料に頼らずに再利用可能エネルギーのみを使用するため、船体を無限に動かすことができます。燃料も船員も必要なければ、環境破壊につながる排ガスもありません。

James Gosling氏がLiquid Roboticsでチーフ・ソフトウェア・アーキテクトに就任して以来、この異色の航海ロボットのミッションにおいてデータ転送/分析に果たすJavaの役割はますます重要になっています。Gosling氏はSun MicrosystemsでJavaを生み出したコンピュータ・サイエンティストとして知られています。そのGosling氏が現在手がけているのが、Wave Gliderの内蔵ソフトウェアのアーキテクチャ再設計と、リアルタイムの海洋情報に直接アクセスするためのData as a Serviceクラウドの改修です。

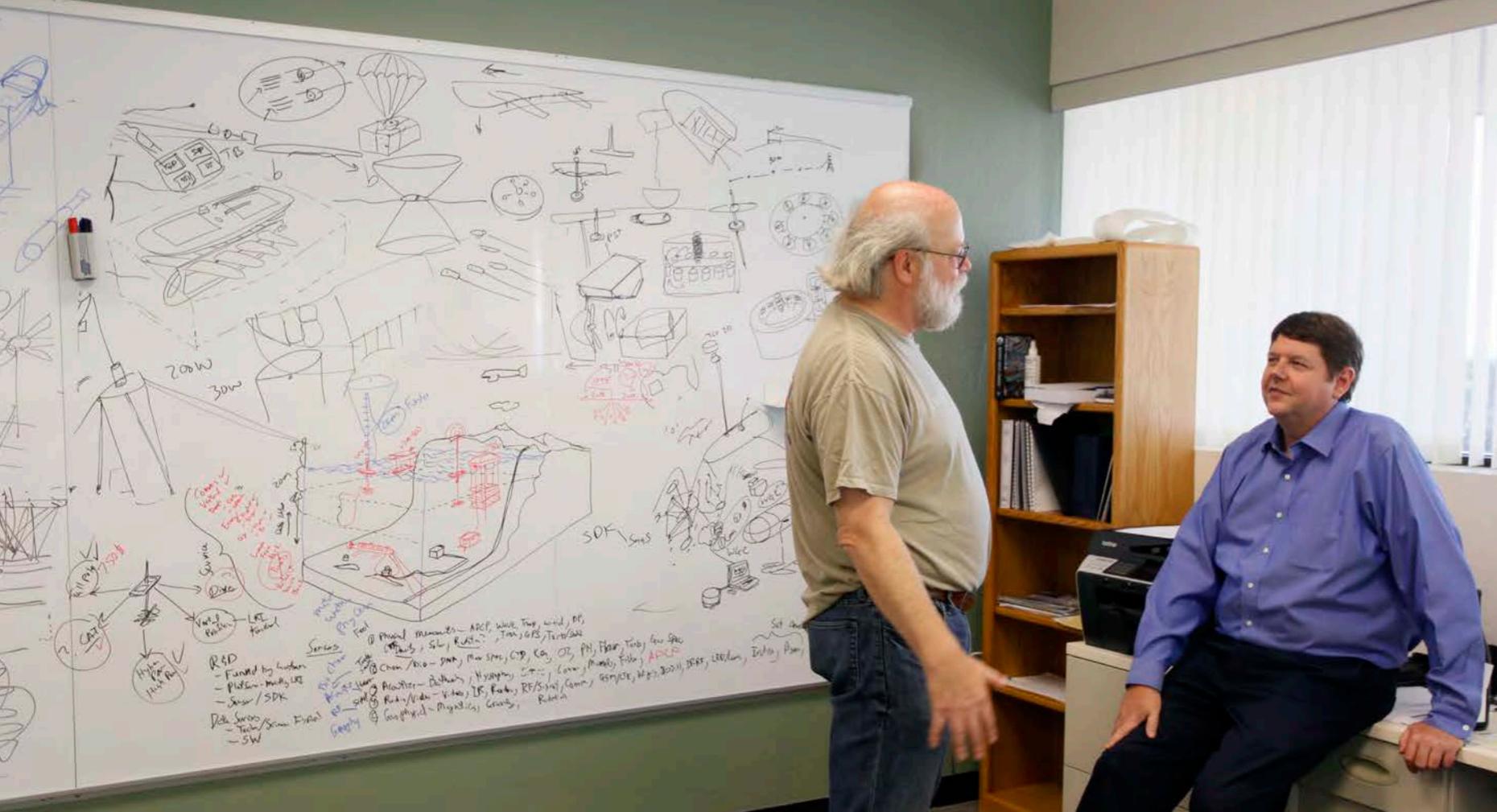
人生の変化

オラクルを離職後、短期間所属したGoogleも離れてまだ間もない2011年9月、Gosling氏は成長著しいLiquid Roboticsに入社しました。Liquid Roboticsについては、

長年の友人で仕事仲間でもあるBill Vass氏(現Liquid Robotics社長兼CEO)から話を聞いていました。

Gosling氏は当時のことを次のように語ります。「Billが立ち上げた新しいベンチャー企業を見に行ったのですが、それはもう感動しました。多岐に渡る詳細な海洋データを、もっとも安く、かつ幅広く取得する方法を生み出していたのですから。Wave Gliderにはデータに関連する興味深い問題や大規模制御の問題がありますが、どちらもまさに私が長年情熱を注ぎ続けている問題です。だから私はBillに、私もここで働けないかと尋ねました」

Vass氏はすぐにGosling氏を雇い、さまざまな挑戦的プロジェクトに自由に取り組んでもらいました。現在Gosling氏はチーフ・ソフトウェア・アーキテクトとして、多数の命令/制御システムの刷新とネットワーク通信の改善に取り組んでいます。このネットワーク通信は、Wave Gliderの各ミッションで得たデータをクラウドベースのデータ・リポジトリにアップロードするためのものです。また、Wave



Gliderの各ミッションの追跡、監視、制御を行うデータ視覚化システムも構築しています。

えます。Gosling氏は次のように説明します。「太平洋をゆっくりと歩くように横断している様子を想像してみてください。それがWave Gliderの仕事です」

平均1~2ノット(1ノット=約0.5メートル/秒)という速度で海を渡ることは、退屈に思えるかもしれませんが、この方法以外で安定した監視システムを配備するには、監視システムを目的地まで運ぶ外航船を借りるしかありません。これには1日あたり5万米ドル~15万米ドルのコストがかかります。おとぎ話の「ウサギとカメ」のように、Wave Gliderは、歩みはゆっくりでも信頼できて疲れを知らず無尽蔵のエネルギー源を自由に使用できます。実際のところ、「ゆっくり」は科学計器には最適です。高速の航行では集めきれない微細な情報も漏らすことなく、密度の高いデータセットを丁寧に収集できるからです。

また、ゆっくり確実に進む方法は、コストの

「当社のプラットフォームの大部分がソフトウェアに頼っています。Wave Gliderはいわば海に浮かぶデータセンターです。多くのインテリジェンスが組み込まれており、このインテリジェンスは衛星経由でクラウドに接続されます」(Vass氏)

現代のおとぎ話

Wave Gliderは、1年もの期間、数千マイルに及ぶようなミッションでも、海洋データを継続的に収集して転送できます。今日までにWave Glider

が世界中を航海した総距離は25万海里を超

海に浮かぶデータセンター

「当社のプラットフォームの大部分がソフトウェアに頼っています。Wave Gliderはいわば海に浮かぶデータセンターです。多くのインテリジェンスが組み込まれており、このインテリジェンスは衛星経由でクラウドに接続されます」
—Bill Vass氏、Liquid Robotics社長兼CEO

JavaによるWave Glider内蔵ソフトウェアの再構築と海洋情報にリアルタイムでアクセスするData as a Serviceクラウド開発の最新状況について、Vass氏に報告するGosling氏 cloud for real-time access to ocean information.

面でも優れています。Wave Gliderの優れたコスト性は米国海洋大気庁、ウッズホール海洋研究所、モンレー湾水族館研究所、スクリップス海洋研究所、ハワイ大学といった名高い研究機関の注目を浴び、現在ではこれらすべての機関が海洋学の実験にWave Gliderを利用しています。

「深海調査用のブイは非常にコストが高く、費用のほとんどは海中でのブイの配備とメンテナンスに充てられます。しかし、Wave Gliderでは、特定のグライダーのメンテナンスが必要になった場合は新しいグライダーを現場に送って交換し、古いグライダーはそのまま陸地まで航海させて戻すことができます」(Gosling氏)

Wave Gliderにはさまざまな利用法があり、気象観測はほんの一例に過ぎません。海底油田/ガス調査などの産業や、国防目的、さらには北極の水溫観測や海中の炭素レベル測定などの環境事業でも採用されています。Wave Gliderの内蔵計器で収集された情報は、科学者が海流、ハリケーン予測、油流出の検知、海洋生物の調査など、多様な用途のデータを収集するために役立ちます。さらに、太陽電池が搭載されており、コンピュータや航行システム、また船体と陸地とをつなぐイリジウム通信用モデムのための発電を行っています。

現場の取り組み

Gosling氏はコーディングに関わることでできる小さな開発チームに戻ったことを嬉しく

ゆっくり確実に

「太平洋をゆっくりと歩くように横断している様子を想像してみてください。それがWave Gliderの仕事です」

—James Gosling氏、Liquid Robotics チーフ・ソフトウェア・アーキテクト

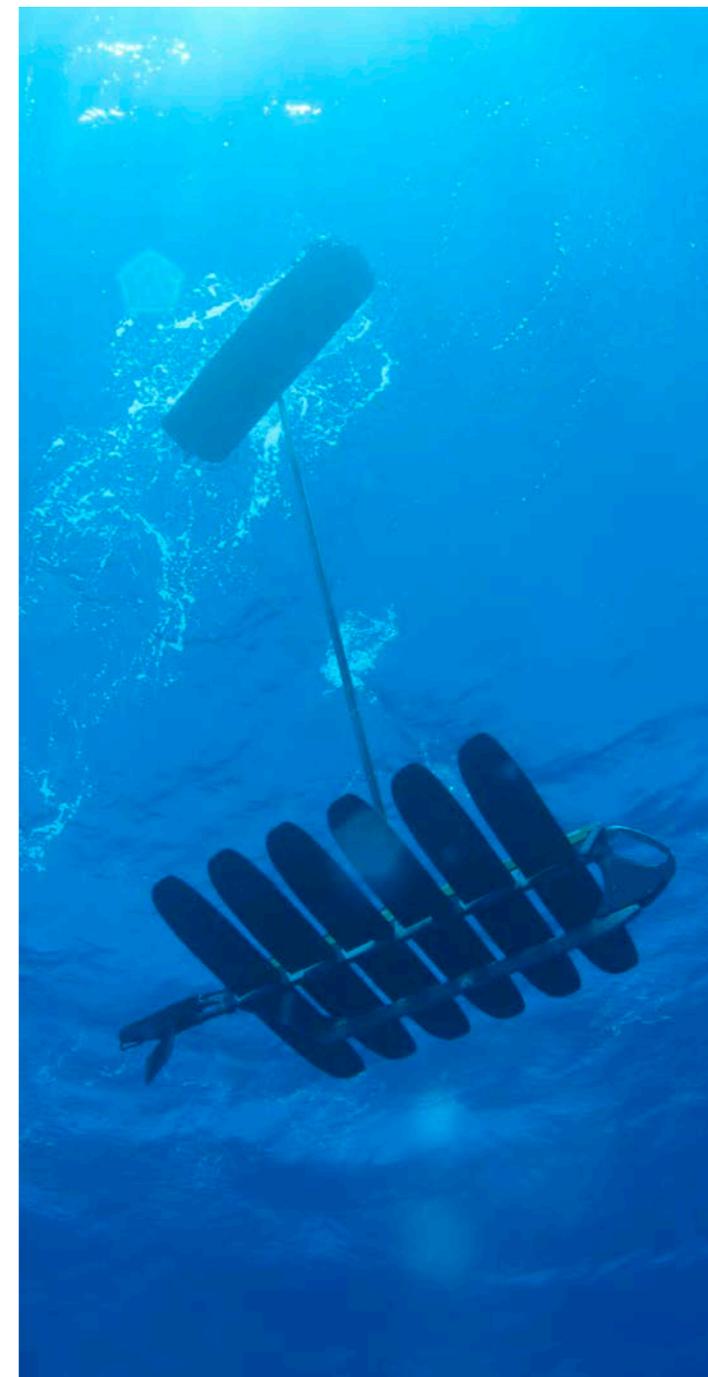
思っています。既存のWave Glider内蔵ソフトウェアの大部分はCで記述されており、小さなマイクロコントローラ上で稼働します。この内蔵ソフトウェアのフレームワークは、一部のケースで限界まで複雑化していました。そのため、Gosling氏とチーム・メンバーは現在、

命令/制御コンポーネントをJavaで体系的に書き直し、最新型のARMプロセッサ上で稼働させる取り組みを行っています。

さらに、Gosling氏はJavaを使用して、顧客のアプリケーションを統合するためのフレームワークを提供するData as a Serviceクラウドを開発しています。Gosling氏いわく、「既存のCプログラムを高度な操縦と移動が可能な洗練されたアプリケーションに改良することは困難ですが、Javaではもっと柔軟に開発できます」

Gosling氏は、Wave Gliderの現在の活動を顧客がデスクトップで確認するための視覚化ソフトウェアも開発しています。この開発ではJDK 7を使用し、Swing GUIウィジェット・ツールキットのコンポーネントやNASA World Windの地図コンポーネントを取り入れています。

Java Foundation Classesに含まれるSwingは、柔軟なGUIを作成するためのAPIです。World WindはNASAが開発するオープンソースの地図インタフェースであり、Google Earth同様、パソコン上で利用するオープンソース・コミュニティでもあります。



写真提供: LIQUID ROBOTICS

Vass 氏と Gosling 氏、Wave Glider の製造エリアにて (左上)。Wave Glider の操縦室で Liquid Robotics のチーム・メンバーと話す Vass 氏 (左中央)。Vass 氏と Gosling 氏、Liquid Robotics の本社にて。後方のスクリーンには Wave Glider の活動状況とデータが映し出されている (左下)。海中から見た Wave Glider (右)。



さまざまな用途

科学 / 環境：大気に接しながら海面を航行する Liquid Robotics の Wave Glider は、気象データと海象データを収集できます。さらに、海中のセンサーを使用したリアルタイムの遠隔測定も可能です。

産業 / 工業：Wave Glider は、調査、産出、長期的監視などのあらゆる段階で、海上エネルギー・プロジェクトをサポートできます。

政治 / 防衛：Wave Glider は、港湾セキュリティから常時監視、海中と空中とのリアルタイム通信まで、広範な用途に利用できます。

は、その非常に優れたパフォーマンスと信頼性にあります。Javaはクラウドベースのコンポーネントで使用するのに適しており、さらに複雑なデータ構造を処理できるため遠隔測定データの分析にも理想的だと、Gosling氏は語ります。「当社のアプリケーションはWebアプリケーションではありません。したがって、HTMLコードの生成は問題ではありません。重要なのは、リモートの情報源にあるデータの自動転送と分析です」

外洋からの衛星通信には法外なコストがかかる



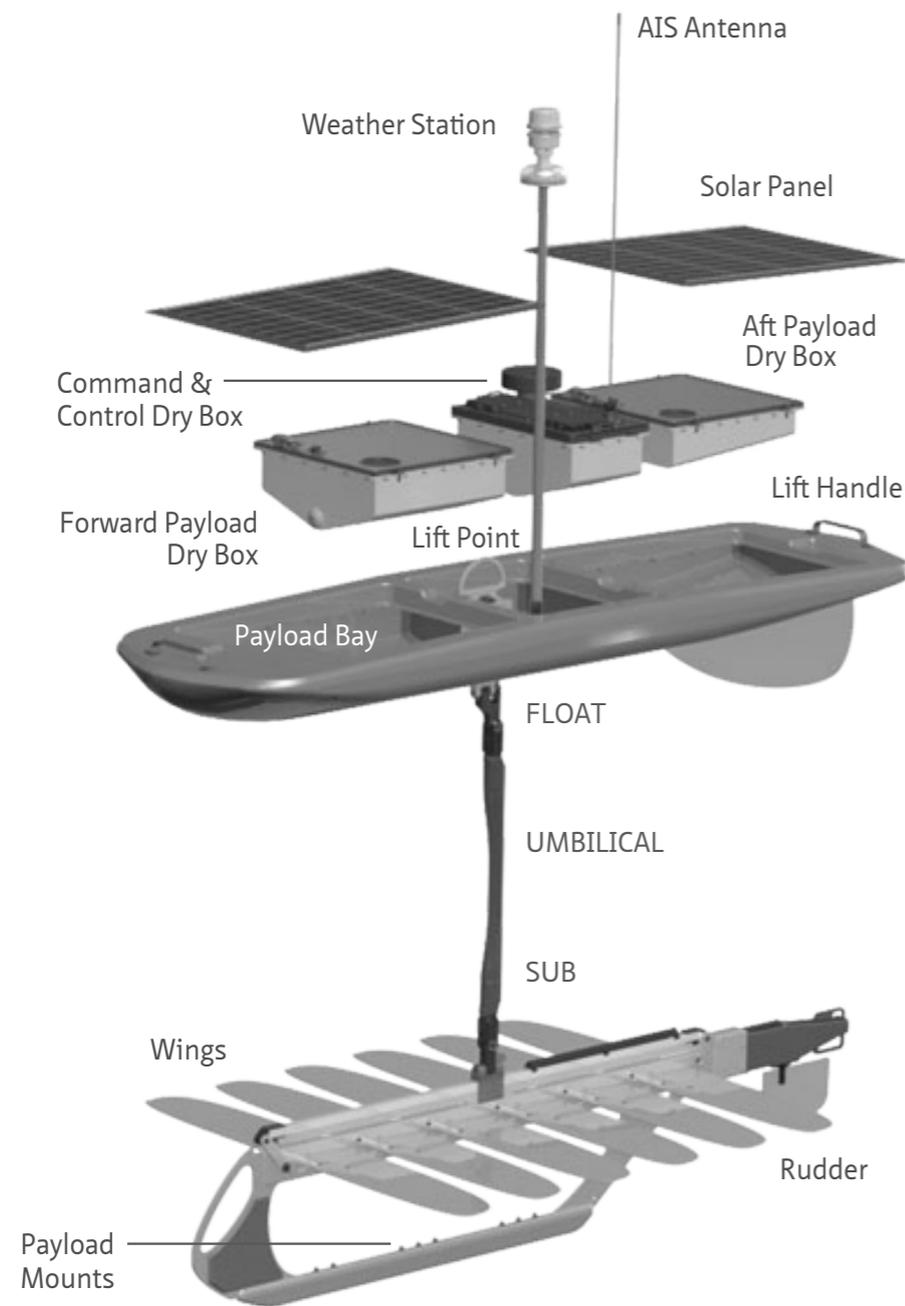
7月に開催されたBay Area All JUG Eventで「Robots and Water and Whales, Oh My! (ロボットと海とクジラ、えっ!)」のプレゼンテーションを行うGosling氏

視覚化ソフトウェアではNASAおよびアメリカ地質調査所の衛星画像や航空写真と地形図を重ね合わせ、世界中の海を漂うWave Gliderを顧客が追跡できるようにしています。

「この新しいインターフェースにはさまざまな計器から読み出した情報が表示され、時系列を前後に切り替えて、Wave Gliderの各ミッションの過程を再現できます。また、複数の船体から取得した遠隔測定データを取りまとめ、近隣にいる船の航路を確認できるように設計されています。そのため、Wave Gliderが航路帯を通過する必要がある場合には、タイムラインを示すアニメーションを表示し、必要に応じてWave Gliderの航路を修正できます」(Gosling氏)

World Windの地図を利用したデスクトップ・インターフェースでは、航路計画から日々のパフォーマンス監視、問題発生時の状況の確認まで、Wave Gliderのミッションの各局面をアニメーションで見ることができます。しかし、航行の大部分で、Wave Gliderは自律的に動作します。Gosling氏は次のように説明します。「Wave Gliderはウェイポイント(経由地点)を辿りながら進み、異常が発生しない限りは、ただ自分の仕事を続けます。しかし、時には海流の激しい領域を通過することや、航路に他の船がいてそのままでは衝突してしまうようなこともあります。その場合はWave Gliderから操縦者のiPhoneにメッセージが送信され、操縦者が別の航路を手動で設定できるようになっています」

なぜJAVAか
Gosling氏がJavaを好んで使用する理由



Wave Gliderの基本構成を表した図。本質的に「浮かぶ台車」と言えるWave Gliderは、太陽電池で覆われ、操縦システム、観測機器制御システム、衛星通信用のシステムが搭載されている。

写真提供: LIQUID ROBOTICS

かるため、Gosling氏は他のプログラミング領域ではほぼ消え去った「帯域幅が非常に限られている」という問題と格闘しています。衛星リンクを通じて送信されるすべてのデータを密にエンコードし圧縮しなければなりません。

「当社のサービスとテクノロジーはネットワーク・コストで動いています。通常、船体と陸地とのデータ伝送は衛星経由で実行されますが、この伝送には1キロバイトあたり約1ドルのコストがかかります。より安価で高速なネットワークを使用することもあります。全体的に見て、海洋上では主にイリジウム通信を使用しています。そのため、当社で1テラバイトのデータを生成するためには、10億ドルのネットワーク・コストがかかります」(Gosling氏)

この問題は、船体が陸地近くにあるときや、広域通信ハブを持つ掘削装置の近辺にあるときは緩和されます。また、データがすぐに必要でない場合は、データを内蔵キャッシュに保管し、後でダウンロードできるため、影響はありません。しかし、ほとんどのデータ伝送/分析機能で、経済性の考慮は不可欠です。

Wave Glider開発の大部分でJDK 7を利用するGosling氏は、Javaデスクトップ・アプリケーションのフレームワークであるNetBeansのヘビー・ユーザーでもあります。現在は、JDK 8 (Java SE 8のプロトタイプ・リファレンス実装)を利用できる日を心待ちにしています。特に関心があるのがProject Lambdaです。

「水質分析でもクジラの鳴き声の観測でも、Wave Gliderはあらゆる種類の検査/測定用計器に対応するプラット

フォームとなります。どのような計器でも、内蔵コンピュータに接続して利用できます」(Gosling氏) </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。

MAKE THE
FUTURE
JAVA

Join the Java Development Team

WE'RE HIRING!

oracle.com/javajobs

ORACLE®





画像:I-HUA CHEN

祝10周年 DUKE'S CHOICE AWARDS

初のCommunity Choice Award受賞者をはじめ、2組のユーザー・グループや国連の難民機関も名を連ねる、本年の多様な受賞者をご紹介します。

PHILIP GILL

本年のDuke's Choice Awardsの受賞者を一見すると、Javaを扱うという点を除いては関連性のない、多様なオープンソース・プロジェクト、企業、ユーザー・グループ、個人が名を連ねているように思えます。これはたしかに事実です。しかし、Javaコミュニティのオスカーとも言えるDuke's Choice Awardsの本年の受賞者をよく見ていけば、Javaの最新の技術的イノベーションがわかるだけでなく、長年の間にJavaコミュニティがいかに成長し変化してきたかについても知ることができます。

Duke's Choice Awardsの受賞者はこれまで技術的なイノベーションに基づいて選出されてきましたが、本年、審査員は受賞の条件を広げることを決定しました。その結果、Duke's Choice Awardsの10年に及ぶ歴史の中で初めて、2組のJavaユーザー・グループ(JUG)が栄冠を手に入れました。このようなユーザー・グループの受賞は、Javaがべ

2012年DUKE'S CHOICE AWARDS 受賞者

(組織名称のアルファベット順)

[AgroSense](#)

[Hadoop](#)

[Apache Software Foundation](#)

[JDuchess](#)

[Jelastic, Inc.](#)

[Liquid Robotics](#)

[London Java Community](#)

[MICE](#)

[北大西洋条約機構\(NATO\)](#)

[Parleys.com](#)

[Ram Kashyap](#)

[Student Nokia Developer Group](#)

[Level One Registration Tool](#)

[国連難民高等弁務官事務所
\(UNHCR\)](#)





JDUCHESS

「私たちは女性同士がつながり合い、この素晴らしいJavaコミュニティに参加できるようにするためのプラットフォームを提供しています」

—Régina ten Bruggencate氏

ンダー主導ではなくユーザー主導の言語であり続けていることを強く示しています。

「本年は、今までよりも自由な決定ができました」と語るのは、本年の審査員の一人であるJohn Yeary氏です。Yeary氏は、サウスカロライナ州グリーンヴィルのJavaユーザー・グループ(GreenJUG)でリーダーを務めています。「総合的に見て、受賞した2組のグループは、技術面でのイノベーションが優れていただけでなく、革新的な手法でコミュニティを発展させたという点が評価されました」

Yeary氏によると、受賞した2組のユーザー・グループはコードの開発自体は行っていないものの、開発者の成長と育成に大きく貢献しています。**London Java Community (LJC)** および同コミュニティのユーザーは、OpenJDKやJava Community Process(JCP)をはじめとする取り組みに積極的に関わってきました。本号ではLJC代表のBen Evans氏の詳しいインタビューも掲載しています。12ページの「JCP Executiveシリーズ:Ben Evansに聞く」をご覧ください。

写真: TON HENDRIKS、JOHN BLYTHE

大半のJUGが特定地域に絞って活動する中で、**JDuchess**はJavaコミュニティへの女性の参加を世界規模で促進しています。共同創業者のLinda van der Pal氏と共にJduchessを運営するRégina ten Bruggencate氏によると、所属するメンバーは60か国500人以上に上ります。

「私たちは女性同士がつながり合い、この素晴らしいJavaコミュニティに参加できるようにするためのプラットフォームを提供しています。開発者、アーキテクト、テスター、ビジネスアナリスト、マネージャーなど、Javaテクノロジーに関心のあるすべての女性を歓迎します。ITによってグローバル経済が進化する中、女性はITの未来で重要な役割を担うと信じて

います」(ten Bruggencate氏)

HEART AND SOUL

Javaが「ビジネスに向いている」ことは疑いようありませんが、本年はJavaテクノロジーを利用して現代の人道的課題や環境問題に取り組む団体が受賞者となりました。**国連難民高等弁務官事務所 (UNHCR)** は、内戦から自然災害まで、世界中の危機の最前線に立って活動しています。人道支援という使命を果たすために、UNHCRは

LONDON JAVA COMMUNITY

「私は個人的に、London Java Communityがコミュニティでの存在感を高めることを望んでいました。実際に参加して、Javaプラットフォームの潜在能力をより多く引き出すことに貢献したかったです」

—Ben Evans氏





UNHCR

「Level Oneアプリケーションによって、
難民への初期支援と対応改善計画の策定
に必要な時間を短縮できました」

—Stanyslas Matayo氏 (写真で立っ
ている人物、チーム・メンバーと共に)

NetBeansプラットフォームを基盤とするJavaシンクライアント・アプリケーションを開発しました。

Level Oneと名付けられたこの登録ツールは、今年、西アフリカのマリ共和国とニジェール共和国で初めて導入されました。Level Oneでは難民の人数に関する情報を収集し、水、食料、住居、医療といった現場での支援の配布計画を立てることができます。さらに、これらの機能ではさまざまな情報源の地理座標情報が活用されます。これによりUNHCRは、適切な内容と量の支援を、必要

としている場所に届けることができます。Level One開発プロジェクトのチーム・リーダーであるStanyslas Matayo氏は次のように説明します。「Level Oneアプリケーションによって、難民への初期支援と対応改善計画の策定に必要な時間を短縮できました」

「Level Oneの登録情報は、家族や友人の登録と検索の問題を解決するために役立ちます。このソフトウェアによる人道的問題への効果ははっきりと表れています。仲間を助けることほど尊いサービスはありません。このようなプロジェクトで活動している人たちがいると思うと、夜も安心して眠ることができます」(GreenJUG Yearly氏)

飢えた世界に食料を届けるために農業の手法を改善すること。こ

れがオープンソースの農場情報管理システムAgroSenseの目標です。**AgroSense**は、JavaおよびNetBeansプラットフォームで構築されています。主要開発者のTimon Veenstra氏によると、「AgroSenseは、農業従事者、農業関連産業、仕入先などが簡単に情報を交換できるアプリケーション・モジュールを開発できるようにするシステ

Community Choice: NATOのMICE

史上初のCommunity Choice Awardは、北大西洋条約機構(NATO)で使用され

ている**MASE Integrated Console Environment(MICE)**に贈られました。MICEは防空作戦および戦場作戦を遂行するための高パフォーマンス視覚化環境であり、NetBeansプラットフォーム上にJavaで構築されています。

NATOプログラミング・センターのソフトウェア・アーキテクトであるAngelo D'Agnano氏によると、「MICEはNATOのMulti-AEGIS Site Emulator(MASE)コンソールの改訂版であり、MASE自体は1990年代にC/C++とMotifを使用して開発されました。MICEでJavaが採用された理由は、NATOの防空指揮統制システム(ACCS)の多くのコンポーネントでJavaがすでに利用されていたためです」

MICEは2つの外部ライブラリに依存しています。1つはパフォーマンスに優れた視覚化ソフトウェア・コンポーネント・セットの「LuciadMap」で、システムで地域情報を表示するために使用されます。もう1つは、アプリケーション・フレームワーク用のNetBeansプラットフォームです。D'Agnano氏は、開発者向けサイトDzoneのNetBeans Zoneに投稿した記事で次のように述べています。「LuciadMapを利用することで、表示に必要なリアルタイム・パフォーマンスを達成できました。一方、NetBeans RCP[リッチクライアント・プラットフォーム]を利用することで、開発チームは膨大な時間を節約できました。安定的で一貫したアプリケーション設計につながるソリューションやデザイン・パターン、ガイドラインをすぐに利用できたからです」

2012年5月、NATOプログラミング・センターは20か国60の地域にMICEコンソールの初期運用版を配備しました。

ムです。この情報交換では、共通基盤としてNetBeansフレームワークを使用します。開発されたアプリケーション・モジュールは、AppleのiTunesのようなオンライン・アプリケーション・ストアで販売されます。非営利のモジュールは農業従事者に無償で提供できます。たとえば、精密農業サポート・モジュールを利用すれば、農業従事者は最小限の肥料や農

写真提供: UNHCR



AGROSENSE

「AgroSenseは、農業従事者が農場にいるときや通信状態が悪いときにはオフラインで作業し、ブロードバンド接続が可能ときには常時接続できるリッチ・クライアントを提供しています」

—Timon Veenstra氏

薬で最大の収穫を得られるようになります」

Veenstra氏は、今ある競合ソリューションは「世界の他の地域と通信しない」か、もしくは農村地域で利用できるとは限らない高帯域幅のインターネット接続を必要としていると話します。「AgroSenseは、農業従事者が農場にいるときや通信状態が悪いときにはオフラインで作業し、ブロードバンド接続が可能ときには常時接続できるリッチ・クライアントを提供しています」

同じくDuke's Choice Awardsを受賞したLiquid Roboticsは、海洋データ・サービス・プロバイダです。同社のテクノロジーであるWave Gliderは、政治、科学、産業の各分野で使用される海洋情報を世界中から収集します。Liquid Roboticsのチーフ・ソフトウェア・アーキテクトを務めるのは、JavaのパイオニアであるJames Gosling氏です。Liquid Roboticsについては、本号17ページの特集記事「海に浮かぶJava」で取り上げています。

テクノロジー・リーダー

驚くまでもなく、本年の受賞者の顔ぶれからは、Javaが現代のテクノロジーにおける最重要課題に第一線で立ち向かっていることもわかります。既存のJavaアプリケーションをクラウドに移行することは手ごわい作業です。しかし、カリフォルニア州パロアルトで新興企業Jelastic, Inc.のチーフ・エバンジェリストを務めるJudah Johns氏によると、同社は、コードの変更もロックインもなく既存のJavaアプリケーションをクラウド内で稼働できる初のJava Platform as a Service(PaaS)を提供しています。さらに、Jelasticのプラットフォームはスケーラビリティに限界がなく、このスケーラビリティの実現においても既存のコードを変更する必要はありません。

「Jelasticプラットフォームは、世界で初め

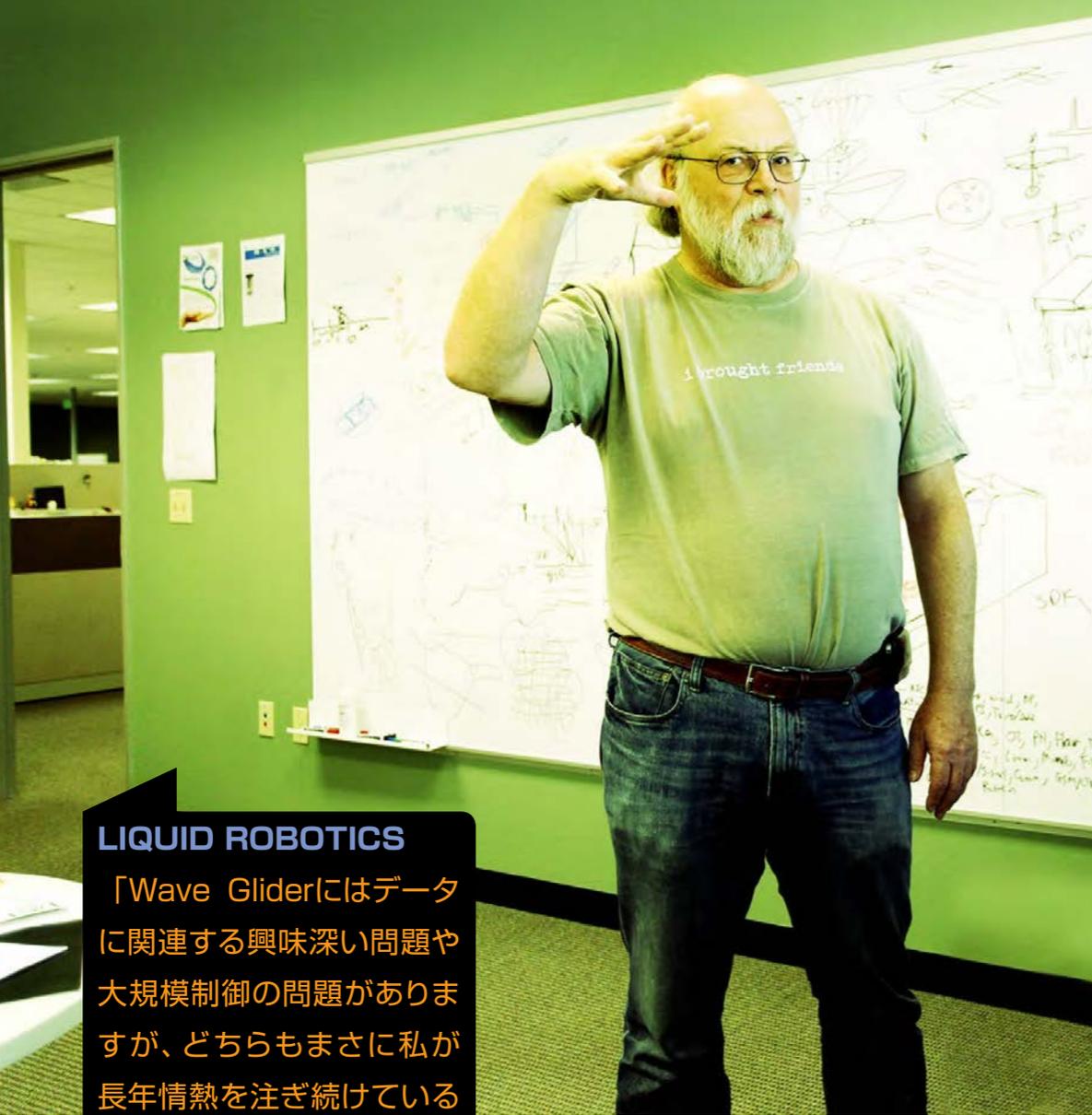
てサービス・プロバイダ経由で提供される、唯一の標準ベースJavaホスティング・プラットフォームです」(Johns氏)

ビッグ・データ(従来型のデータ管理ツールでは扱いきれない非常に大規模で複雑なデータ)の処理は、あらゆる組織で急務となっています。Apache Software FoundationのHadoopプロジェクトは、Javaを開発言語とし、数台のサーバーから数千台のマシンまで、複数のコンピュータによるクラスタ構成全体で大規模なデータ・セットを分散処理するためのフレームワークを提供します。

Apache Software Foundationの会長でありHadoopプロジェクトの創設者でもあるDoug Cutting氏は次のように説明します。「Hadoopを使用する組織は、作成したデータからより高い価値を引き出すことができます。Hadoopがあれば、以前は破棄していたデータも保管して分析できます。より多くのデータを活用することで、ビジネスをより深く理解し、改善することができるのです」

Cutting氏はさらにこう述べます。「JavaはHadoopエコシステムの主要言語であり、Hadoopはビッグ・データ向けオペレーティング・システムの事実上の業界標準となっています。そのため、ビッグ・データを重視する傾向が広がるほど、Javaの利用も広がります」

さらに、既存のテクノロジーに新しい工夫を加えた2組が受賞者となりました。ベルギーのブルッヘを拠点とするEラーニング専門企業のParleys.comは、Javaテクノロジーを利用して、オンライン講義と多様なITカンファレンスをデスクトップ、ラップトップ、iPad、Android端末、BlackBerry PlayBook向けに配信しています。ParleysはDevovxやJavaOneを含む1,700以上のカンファレンスを配信し、その一意訪問者数は80万人を超えていると創設者兼CEO



LIQUID ROBOTICS

「Wave Gliderにはデータに関連する興味深い問題や大規模制御の問題がありますが、どちらもまさに私が長年情熱を注ぎ続けている問題です」

—James Gosling氏

写真: BOB ADLER

のStephan Janssen氏は説明します。Janssen氏によると、「Parleysのユーザーは、プレゼンテーションをデスクトップで視聴することも、端末にダウンロードして時と場所を問わず好きなときにオフラインで楽しむこともできます」

最後に、今年の学生受賞者である**Ram Kashyap**氏をご紹介します。Kashyap氏は過去にJava Magazineに登場しています。March/April 2012号特集記事「[The New Java Developers](#)」で、学生グループのStudent Nokia Developerの創設者兼代表者として紹介されました。その後もKashyap氏は多忙な日々を送っています。インドのバンガロールにあるピープルズ・エデュケーション・ソサエティ工科大学を卒業し、Javaモバイル事業の立ち上げに取り組みながら、学生を対象にJava MEのトレーニングも行っています。

</article>

—**Philip Gill**、カリフォルニア州サンディエゴを活動拠点とするライター兼編集者。20年もの間Javaの動向を追い続けている

審査員と 審査過程

10年目を数える今年のDuke's Choice Awardsでは、受賞者の選出を3段階に分けて行いました。まず、Java.netのすべてのコミュニティ・メンバーに対し、今年の審査対象候補の推薦を募りました。審査の第1段階と第2段階で、審査員は、推薦された68組の候補から9組の受賞者を選出し、新たに創設されたCommunity Choice Awardに6組の候補を推薦しました。最終段階では、Community Choice Awardの候補者をJava.netに公表し、このJavaコミュニティの全メンバーから、同賞にもっともふさわしいと思う候補者への投票を募りました。最多得票のNATOプログラミング・センターのMICEプロジェクトは、41パーセントの投票を獲得しました。

今年の審査員は次のとおりです(敬称略)

Yara Senger、SouJava

John Yeary、グリーンヴィルJavaユーザー・グループの代表者兼創設者

Glen Peterson、グリーンヴィルJavaユーザー・グループのメンバー/
PlanBase Inc CTO

Martijn Verburg、ロンドンJavaコミュニティ(同氏はロンドンJavaコミュニティへの投票を棄権しました)

Michelle Kovac、Javaマーケティング/運用担当

Arun Gupta、Javaエバンジェリスト/
GlassFishコミュニティのメンバー

Sharat Chander、Javaエバンジェリスト・チーム・マネージャー

10組の受賞者は、9月30日から10月4日までカリフォルニア州サンフランシスコで開催されるJavaOneで表彰されます。



MICHAEL KÖLLING



BIO

パート2

BlueJを使用したオブジェクトの相互作用に関する学習

オブジェクト指向プログラミングにおける抽象化とモジュール化

前号の Java Magazine では、BlueJ 環境について概要を簡単に説明し、初心者にはプログラミングの概念を教えるためのさまざまな BlueJ 付属ツールを紹介しました。BlueJ を使用することにより、理解しづらいプログラミングの基本概念も理解しやすくなります。本記事ではこの点について、小さなサンプル・プログラムを使用して説明します。

今回も、ダウンロードして実際に操作できる実践的なサンプル・プログラムを使用します。

時計のサンプル

オブジェクトの相互作用を説明するために本記事で使用するプロジェクトは、デジタル時計の文字盤です。この文字盤には、「時」と「分」がコロンで区切られて表示されます (図 1)。本記事の演習では、まずヨーロッパ式の 24 時間制時計の文字盤を作成します。すなわち、00:00 (午前 0 時) から 23:59 (午前 0 時の 1 分前) までの時間が表示されます。詳しい調査により、

12 時間制の時計の作成は 24 時間制よりも少し難しいことがわかっていました。そのため、12 時間制の時計については本記事の最後に取り上げます。

抽象化とモジュール化

プログラミングの初心者の多くがまず考えつくことは、1 つのクラスの中で時計の文字盤全体を実装するということです。この発想の中心には、「クラスは何のためにあるか」という、クラスの役割に関する思考があります。「クラスはものを表し」、「時計はものである」というのがこの発想の根幹です。

しかし、本記事ではこの時計の問題に、少し異なるアプローチで取り組みます。この問題を構成している複数の要素を特定し、個別のクラスに分解できるかどうかを検討します。分解を行う理由は複雑性です。プログラミングの実践が進むにつれて、作成するプログラムは複雑になります。初期の演習に見られる小さなタスクは、それぞれ 1 つの

問題として解決できるものです。1 つのクラスを使用してタスク全体を検討し、解決策を考え出すことができます。しかし、より複雑な問題では、1 つのクラスを使用する方法では単純過ぎます。問題の規模が大きくなるほど、その細部すべてを同時に把握し続けることが難しくなります。

このような複雑性の問題に対処できる解決策が抽象化です。抽象化では、1 つの問題を複数のより小さな問題に分解し、その小さな問題をさらに小さな問題に分解します。このようにして個々の問題が容易に対処できる大きさになるまで分解を続けます。分解された問題の 1 つを解決した後は、その問題の詳細については検討せず、その問題の解決策を、次の問題を解決するための 1 つの構成要素として扱います。この技法は分割統治法と呼ばれることもあります。

ここまでの内容を、例を使用して説明します。ある自動車メーカーのエンジニア・チームが新しい自動車を設計しているとしま

11:03

図 1

す。あるエンジニアは、ボディの外装の形状やエンジンのサイズと取り付け位置、乗車エリアの座席の数やサイズ、車輪間の正確な間隔など、自動車の部品について検討しています。一方、別のエンジニアはエンジン設計を担当し (本来はエンジニアのチーム全体で担当する作業ですが、ここでは例として使用するために少し単純化しています)、シリンダ、噴射装置、気化器、電子回路など、エンジンのさまざまな部品について検討しています。エンジン担当のエンジニアはエンジンを 1 つのエントティ (実体) ではなく、多くの部品で構成される複合的な製作物と考えています。エンジンの部品の例として、スパーク・プラグが挙げられます。

さらに、スパーク・プラグを設計する（おそらく別の会社に勤める）別のエンジニアも存在します。このスパーク・プラグ担当のエンジニアもまた、スパーク・プラグを、多くの部品で構成される複合的な製作物ととらえています。複雑な調査を実施して、接合部に使用する金属の種類や、絶縁体に使用する材料の種類と生産工程をすでに決定していることもあります。

他の多くの部品についても同じことが言えます。製造チェーンの最上流にいる設計者は、車輪を1つの部品として扱います。一方、製造チェーンの下流にいるエンジニアは、タイヤの製造に適した材料を生産するための化学組成について日々考察しています。タイヤのエンジニアにとって、タイヤは複合体です。一方、自動車メーカーは、単純にタイヤ・メーカーからタイヤを購入し、このタイヤを1つの完成されたエンティティとして考えます。これが抽象化です。

自動車メーカーのエンジニアは、タイヤ製造の詳細を抽象化することで、たとえば車輪の構成の詳細に集中できます。自動車のボディ形状をデザインするデザイナーは、車輪やエンジンの技術的な詳細を抽象化して、ボディのデザインに集中します（自動車のボディのデザイナーは、エンジンと車輪のサイズにしか関心がありません）。

他のすべての構成要素についても同じです。車内の乗車スペースのデザインに関係する人もいれば、最終的に座席のカバーに使用される生地の開発に取り組む人もいます。

重要な点は、十分な詳細レベルで見

た場合、1台の自動車はあまりにも多くの部品で構成されているため、一人の担当者が同時にすべての部品のすべての詳細を把握することは不可能だということです。一人ですべてを細部まで把握しなければならないとすれば、自動車は製造できません。

自動車を製造できるのは、エンジニアがモジュール化と抽象化を行っているからです。1台の自動車を独立した複数のモジュール（車輪、エンジン、変速機、座席、ハンドルなど）に分解し、それぞれのモジュールを別々の担当者が担当します。モジュールの作成時には抽象化を行います。作成したモジュールは、より複雑な複数の部品（サブコンポーネント）で構成される1つの部品（コンポーネント）として扱います。

したがって、モジュール化と抽象化は相補関係にあります。モジュール化とは、大きなもの（問題）をより小さな要素に分解するプロセスです。一方、抽象化とは、より広い視野に集中するために詳細を無視する能力です。

ソフトウェアにおけるモジュール化と抽象化

モジュール化と抽象化はソフトウェア開発でも使用されます。複雑なプログラムで全体像を維持するために、独立したエンティティとしてプログラミングできるようなサブコンポーネントの特定を試みます。次に、それらのサブコンポーネントを単純な1つの構成要素であるかのように使用します。サブコンポーネント内部の複雑性については関心を持ちません。

オブジェクト指向プログラミングで

は、これらのコンポーネントおよびサブコンポーネントがオブジェクトにあたります。オブジェクト指向言語を使用してソフトウェア内で自動車を作成する場合には、自動車エンジニアと同じことをプログラマーも行います。すなわち、自動車を1つの完全体としてのオブジェクト内に実装するのではなく、まずエンジン、変速機、車輪、座席などに対応する個別のオブジェクトを作成し、次にそれらのオブジェクトからより大きな自動車オブジェクトを組み立てます。

ある問題に対してソフトウェア・システム内に用意すべきオブジェクト（および対応するクラス）を特定することは、かならずしも容易ではありません。本記事では、BlueJを利用してこのようなコンポーネントの作成、テスト、試行を行う方法について説明します。

それでは、デジタル時計に戻りましょう。

プロジェクトの分析

ダウンロードした時計表示プロジェクトをBlueJで開いてください。最初に、非常に単純なクラス図が表示されます（図2）。このプロジェクトは、`NumberDisplay`と`ClockDisplay`という2つのクラスで構成されています。

`NumberDisplay`は、2桁の数値を1つ表示するためのクラスです（図3）。これ以降、この`NumberDisplay`クラスを使用して時計の文字盤を作成できることを確認していきます。実際には、この数値表示クラスの2つのインスタンスを使用します。それぞれ、「時」に対応するインスタンスと、「分」に対応するインスタンスです。このようにして、時計の文字盤を作成するという問題を、より小さく解決しやすい複数の問題に分解しています。

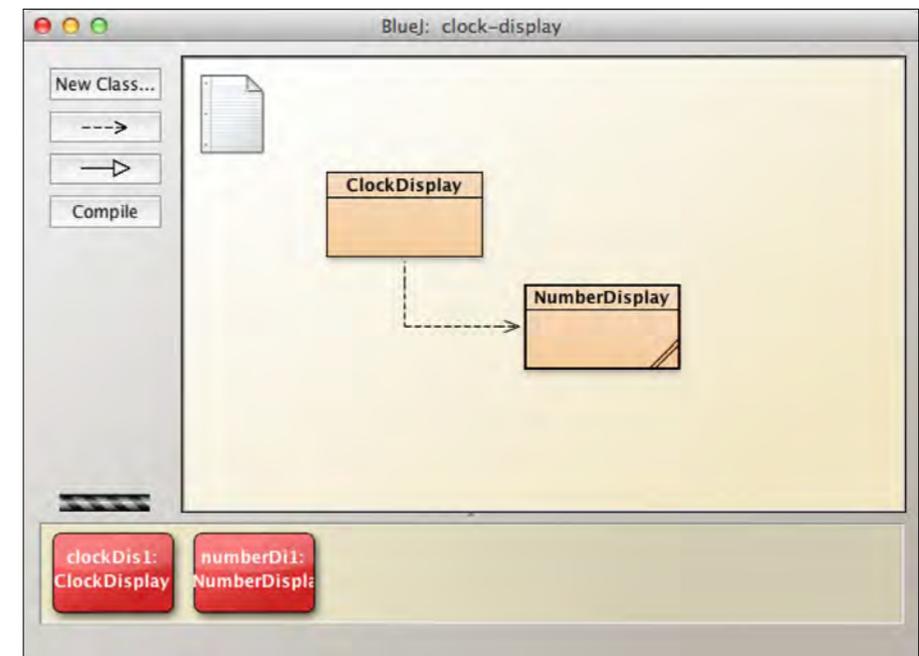


図2

03

図3

まず、2桁の数値を1つ表示するクラスを作成し、次に2つの数値の表示(「時」の表示と「分」の表示)をまとめて時計の文字盤を組み立てます。それぞれの分解された問題は、全体的な1つの問題よりも解決しやすくなっています。

`NumberDisplay` クラスは、指定した上限値までの数値を1つ格納できるオブジェクトを表すクラスです。この数値はインクリメントする(1ずつ増やす)ことができ、上限に達するとゼロに戻ります。`NumberDisplay` オブジェクトの1つを上限60として「分」に使用し、もう1つを上限24として「時」に使用します。

BlueJでは、ソース・コードを読む、またはメソッドを呼び出してオブジェクトの動作を試すという2種類の方法でオブジェクトを分析できます。ここからは、実際にオブジェクトの動作を試す方法でこのプログラムを見ていきます。

まず、クラス図で `NumberDisplay` クラスを右クリックし、メニューの1つ目の項目「`new NumberDisplay(int roll OverLimit)`」を選択して、

`NumberDisplay` クラスのインスタンスを作成します(図4)。`new NumberDisplay(int roll OverLimit)` メニュー・コマンドを使用すると、クラスのコンストラクタが呼び出され、オブジェクトが作成されます。

ダイアログ・ボックスが表示され、インスタンス名とコンストラクタのパラメータの値を入力するように求められます。ここで、表示をゼロに戻すための上限値を指定できます。ここでは、インスタンス名には示された名前をそのまま使用し、パラメータには60を指定します。「OK」をクリックすると、`NumberDisplay` オブジェクトがオブジェクト・ベンチに表示されます(図2)。

この `NumberDisplay` オブジェクトを右クリックし、さまざまなメソッドを実行して `NumberDisplay` の動作を試すことができます(図5)。

`NumberDisplay` オブジェクトには4つのメソッドが含まれています。それぞれのメソッドを選択して、動作を試してください。各メソッドの動作は以下のとおりです。

- `setValue(int replacementValue)`: 表示する値の設定
- `getValue()`: 現在の値を返す
- `getDisplayValue()`: 結果が常に2桁の数値になるようにゼロをパディングした値(例: “4”ではなく“04”)を(`String`で)返す
- `increment()`: 値のインクリメント

`NumberDisplay` オブジェクトの動作を試す別の方法として、オブジェクトのメニューで「**Inspect**」機能を使用してオブジェクト・インスペクタを開くという方法もあります(図6)。

オブジェクト・インスペクタでは、`NumberDisplay` オブジェクトに2つのフィールドがあることがわかります。現在の値を保持するフィールドと、上限値を保持するフィールドです。オブジェクト・インスペクタを開いたままにして、オブジェクトのいくつかのメソッドを再度呼び出してください。値が変化の様子やメソッドの実行結果を観察できます。値が上限間近のときに `increment` メソッドを呼び出すと何が起きるかは興味深い実験です。ぜひ試してみてください。

さらに、従来のクラスの分析方法として、ソース・コードを読むという方法もあります。`NumberDisplay` クラスのソース・コードを開き(クラスアイコンをダブルクリック)、ソース・コードに目を通してください。4つのメソッドの実装方法を確認できます。

ClockDisplay の分析

`NumberDisplay` オブジェクトの動作を実際に試してみることによって、同オブジェクトの振る舞いや機能について感覚をつかむことができました。次に、2つの `NumberDisplay` オブジェクトを使用して時計の文字盤を作成する方法について、`ClockDisplay` クラスを確認していきます。

ここでもまずは、オブジェクトを作成してメソッドの動作を試すことから始めます。

- `ClockDisplay` クラスのオブジェクトの作成: クラスを右クリックし、コンストラクタを選択します。パラメータ付きとパラメータなしの2つのコンストラクタが表示されるため、両

方のコンストラクタの動作を試行してください。

- メソッドの動作の試行: オブジェクト・ベンチで `ClockDisplay` オブジェクトを右クリックしてメソッドを試行します。`getTime`、`setTime`、`timeTick` の3つのメソッドが表示されます。
- `ClockDisplay` オブジェクトのオブ



図4

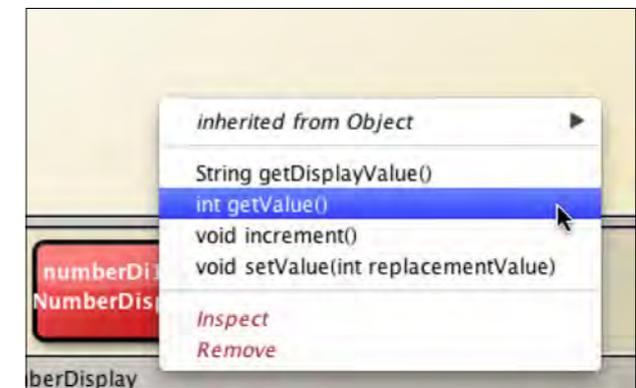


図5

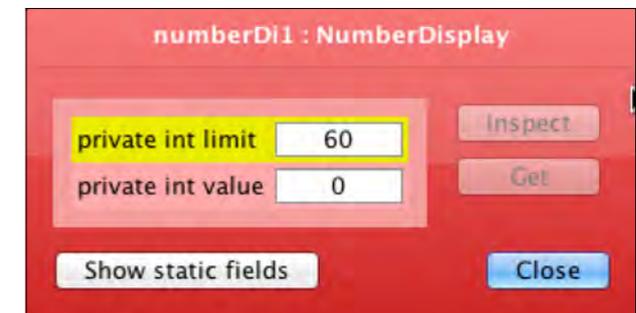


図6

ジェクト・インスペクタを開く (図 7): メニューで「Inspect」を選択し、メソッド呼び出し中にフィールドを観察します。以下のようなさまざまな点を観察できます。

- `ClockDisplay` オブジェクトに、時刻を格納するための `NumberDisplay` 型の 2 つのフィールド (`hours` と `minutes`) があること
- `ClockDisplay` オブジェクトの `displayString` フィールドに時計の現在時刻が表示されること。 `displayString` フィールドをプロジェクトで使用して、時計の表示をシミュレートできます。実際の時計で考えると、`displayString` フィールドは文字盤に表示される時刻に当たります。
- `timeTick` メソッドは何も実行しないように見えること

最後に挙げた `timeTick` メソッドの動作について、メソッドを呼び出しても処理が実行されない原因は単純です。それは、プロジェクトの実装が完了していないからです。実際の時計では、`timeTick` メソッドは時計の時刻を進めるために時計のタイマによって定期的に (1 分に 1 回) 呼び出されます。それでは、この時刻を進める振る舞いを実装していきます。

ソース・コードの分析

まずは `ClockDisplay` のソース・コードを分析します。 `ClockDisplay` クラスのアイコンをダブルクリックして、エディタを開きます。

コードを一通り読むと、面白いことがわかります。3 つのフィールドが宣言されており、そのうち 2 つは

`NumberDisplay` 型、1 つは `String` 型であるということです。

```
private NumberDisplay hours;
private NumberDisplay minutes;
private String displayString;
```

最初の 2 つのフィールドである `hours` と `minutes` は、それぞれ「時」と「分」を表す `NumberDisplay` オブジェクトを保持します。3 つ目の `displayString` フィールドは、時計の現在の時刻表示を表します。クラスのコンストラクタで、この 3 つのフィールドを初期化しています。

```
public ClockDisplay()
{
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    updateDisplay();
}
```

他のメソッドについてもソース・コードを確認し、メソッドの動作について理解してください。これらのメソッドの実装がそれほど複雑ではないことに気づくはずですが、また、`timeTick` メソッドが空であるために時計がまだ動作しないということもわかります。それでは、この `timeTick` メソッドを実装します。

`timeTick` メソッドの目的は、時計の時刻を 1 分進めることです。そのために、以下のコード行をメソッドの本体に追加します。

```
minutes.increment();
```

必要な実装はこれでほぼ完了です

が、1 つだけ足りない点があります。それは、「分」だけでなく「時」も一緒にインクリメントされるケースに対応することです (たとえば、03:59 から 04:00 に進む場合)。このようなケースは、「分」がゼロに戻るときに発生することがわかっているため、以下のコードを追加します。

```
if(minutes.getValue() == 0) {
    hours.increment();
}
```

`hours` のインクリメントを行った後に、表示される値も更新する必要があります。

```
updateDisplay();
```

これで実装はすべて完了です。上記の実装をクラスに追加した後の `timeTick` メソッドはリスト 1 のようになります。

実践

それでは、プログラミングを実際に試してみてください。本記事で説明したとおりにコードを入力し、クラスをコンパイルします (エディタ・ウィンドウまたはメイン・ウィンドウの「Compile」ボタンをクリック)。エラーが発生した場合は、コードを慎重に調べ、原因となる部分を修正します。

次に、`ClockDisplay` オブジェクトを再度作成し、オブジェクト・インスペクタを開いて、オブジェクトのメソッドの動作を試します。特に、新しく実装し



図 7

た `timeTick` メソッドについて試してください。動作するようになったことを確認できます。さらに、「時」が変更される状況も試し、この場合も同様に動作することを確認します。

その他の演習

これまで、BlueJ を使用して既存のプロジェクトを分析する方法、および小さな改修を行う方法について説明しました。プログラミングの練習を積むために、さらに以下のような改修を行うこともできます。

- ユーザーが無効な値を時刻として設定しようとしたときに警告を表示
 - 時計の文字盤に秒を追加
 - ヨーロッパ式の 24 時間制 (16:23) ではなく米国式の 12 時間制 (04:23 p.m.) で表示するように時計を修正
- それぞれのタスクがちょっとしたプログラミングの演習になっており、基本を理解し、プログラミング・スキルの実践の場を求めている初心者にとって最適です。

まとめ

本記事では、小さなサンプル・プログラムを使用して、コードの構成について分析し、その後既存の機能の理解に進み、最終的に新しい機能を追加してプロジェクトを拡張する方法を紹介しました。BlueJを基本的な概念の学習に役立てられることがはっきりとわかりました。

BlueJ環境の以下のような側面は、教師にとっても学習者にとっても、学習プロセスに不可欠です。

- プロジェクトに含まれるクラスやクラス間の関係を容易に確認できる
- 対話型操作によってクラスのインスタンス作成やメソッド呼び出しを行い、クラスを分析できるため、個々のオブジェクトの振る舞いを十分に理解できる
- オブジェクト・インスペクタを使用してオブジェクトの内部を調査できる
- オブジェクト・ベンチを使用して容易にテストし動作を試すことができる
- GUIや**メイン**・メソッドに邪魔されることなくオブジェクトの構造やプログラム・ロジックに集中できるため、面倒な細部の説明に取り組む前に、有用なオブジェクト指向の原則を教えることができる（言うまでもなく、GUIや**メイン**・メソッドも追加可能）

教師の皆さんをはじめ、オブジェクト指向プログラミングを教える立場にある方は、ある考えに辿り着くことでしょう。それは、BlueJが提供する独自のツールセットは、NetBeansやEclipseなどの大規模なプロフェッショ

LISTING 1

```
/**
 * This method should get called once every minute - it
 * makes
 * the clock display go one minute forward.
 */
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) { // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

 [Download all listings in this issue as text](#)

ナル向け環境よりもプログラミングの基本の学習に非常に適しているということです。1年目のプログラマーにちょうど必要となる種類と量のツールが含まれており、開発者はオブジェクト指向の概念を適切に理解した後に、BlueJを卒業してプロフェッショナル向けの環境に進むことができます。

これまでの記事で説明した内容は、BlueJツールの基本に過ぎません。今回は、テストに特化したサポートについて説明します。 **</article>**

LEARN MORE

- [Java SE 7 API](#)
- [BlueJのWebサイト](#)

MAKE THE FUTURE
JAVA



FIND YOUR JUG HERE

One of the most elevating things in the world is to build up a community where you can hang out with your geek friends, educate each other, create values, and give experience to you members.

Csaba Toth
Nashville, TN Java Users' Group (NJUG)

[LEARN MORE](#)



ORACLE®



MAX BONBHEL

BIO

パート2

Webサービスのセキュリティ入門サーバーからクライアントまで

SSLを使用したトランスポート・セキュリティの設定によるWebサービスのセキュリティ向上

全3回シリーズの第1回は、Metro、GlassFish、NetBeans IDEを利用して、サーバーとクライアントの両方でWebサービスのセキュリティを効率的に保護する方法について説明しました。

第2回となる本記事では、NetBeansの各種ツールとGlassFishのデフォルトのデジタル証明書を使用して、トランスポート・セキュリティ機構を設定し、Secure Sockets Layer (SSL) による認証でトランスポート中のAuctionAppアプリケーションを保護する方法を説明します。本記事の目的は、トランスポート層のセキュリティを追加することがいかに簡単であるかを説明することにあります。トランスポート層のセキュリティを追加することにより、セキュリティ保護されたHTTPトランスポート・プロトコルであるHTTPS経由で、SSLを使用してクライアント/サーバー間のデータ転送を保護できるようになります。

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。

トランスポート・セキュリティ機構とは

トランスポート・セキュリティとは、ポイントAとポイントBの間を循環するデータのセキュリティを保護するための機構です。SSLと併用されることが多く、データの機密性に関するさまざまな側面に対応できます。トランスポート・セキュリティを設定することにより、サーバーとクライアントが、証明書の交換によって共通のデータ暗号化アルゴリズムを解読できるようになります。

前提条件

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てください。

- NetBeans IDE 7.2 (ダウンロードは[こちら](#))

- GlassFish 3.1.2.2 (ダウンロードは[こちら](#))
- Metro 1.3以降 (NetBeansに含まれます)
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7.2)を使用しました。

概要: Webサービスに対するトランスポート・セキュリティの追加

本記事では、SSLで保護されたセッションで、クライアントとサーバーの間で送受信されるバイト・データのセキュリティを保護します。

実行するタスクは次のとおりです。

- Webサービスのセキュリティ保護: トランスポート・セキュリティ機構 (SSL) の追加、HTTPSを使用するためのサービスの設定
- Webサービスを参照する新しいクライアントの作成と設定: セキュリティ保護されたWebサービス記述言語 (WSDL) ファイルの指定

- セキュリティ保護されたWebサービスのテスト

注:本シリーズでセキュリティを保護するアプリケーションは、前シリーズ(「[RESTful Webサービスの概要](#)」)で作成したオンライン・オークション・サイト (eBayのようなサイト) です。売主はリストに商品を出品し、買主はその商品に入札します。売主は1つ以上の商品を出品でき、買主は1つ以上の商品に入札できます。

本記事では、入札額を予測するJava API for XML Web Services (JAX-WS) Webサービスへのアクセスを制限します。

Webサービスに対するトランスポート・セキュリティの追加

NetBeans IDEを使用して、Webサービスのセキュリティを保護するためのトランスポート・セキュリティを追加します。この作業は数分で完了します。

1. Transport with Symmetric Keyというセキュリティ機構を



AuctionAppアプリケーションに追加します。

- a. NetBeans IDE 7.2以降でAuctionAppプロジェクトを開きます。
- b. AuctionAppプロジェクトの「**Web Service**」ノードを展開し、「**AuctionAppSOAPws**」ノードを右クリックして、「**Edit Web Service Attributes**」を選択します。
- c. **Quality Of Service**タブにある「**AuctionAppSOAPws PortBinding**」セクションを展開

します(図1参照)。

- d. 「**Reliable Messaging Delivery**」オプションの選択を解除します。
- e. 「**Secure Service**」を選択し、Security Mechanismリストから「**Transport Security (SSL)**」を選択します。
- f. 「**Configure**」をクリックし、「**Require Client Certificate**」オプションを選択します。
- g. 「**Use Development Defaults**」を選択します。
- h. 「**OK**」をクリックします。

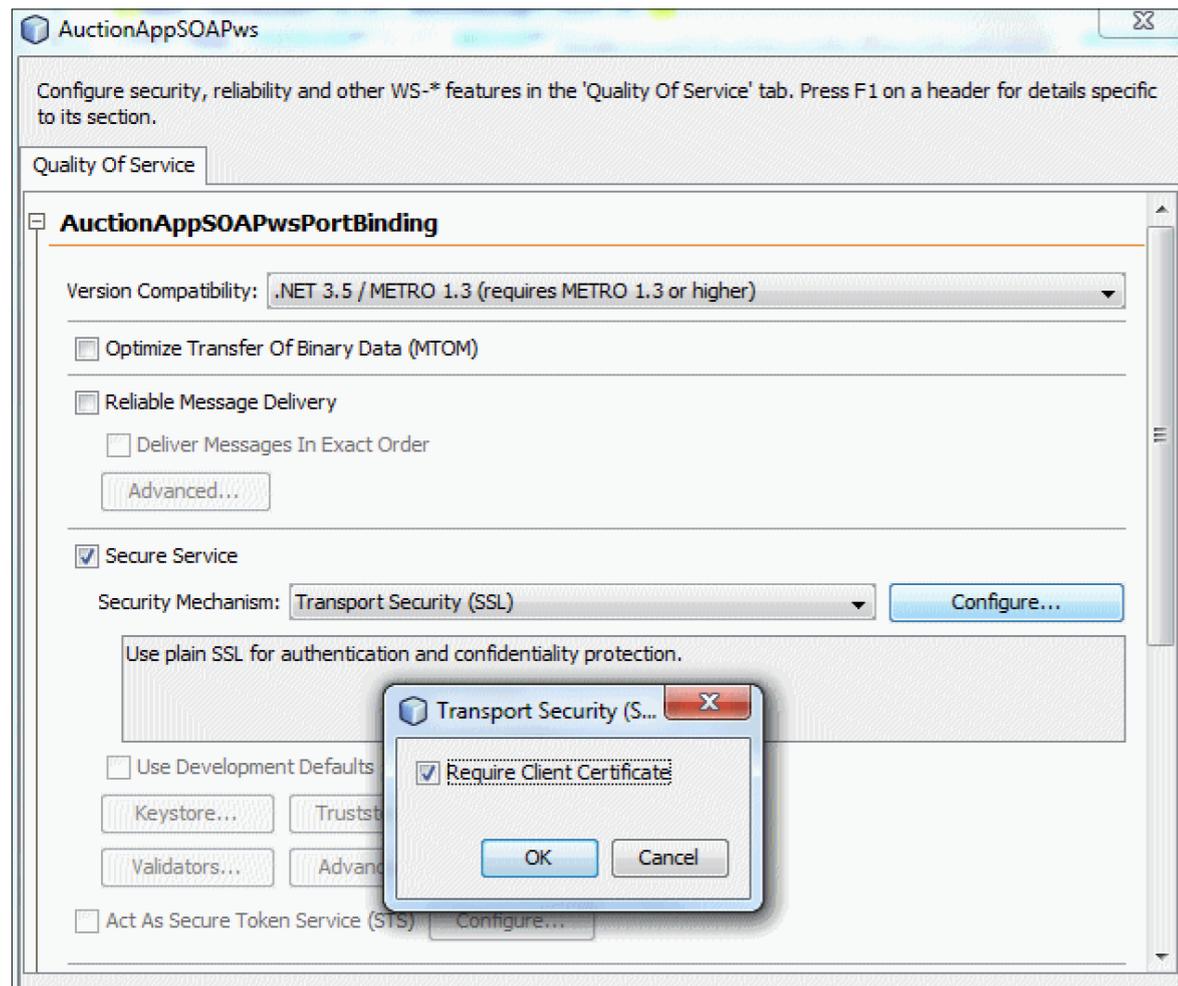


図1

この時点で、セキュリティに関連する要素を含む新しいWeb Services Interoperability Technology (WSIT) 設定が、wsit-com.bonbhel.oracle.auctionApp.AuctionAppSOAPws.xmlファイルのwsp:Policyタグ内部に作成されます。このWSIT設定ファイルは、AuctionAppプロジェクトの「Web Pages/WEB-INF」ノード内にあります。

2. HTTPSを使用するようにサービスを設定します。AuctionAppアプリケーションでSSLが使用されるようにするために、保護するURLのパターンやTransport Guaranteeオプションなどのセキュリティ要件をデプロイメント・ディスクリプタ・ファイル(Web.xml)内に指定します。
 - a. AuctionAppプロジェクトの「**Web Pages/WEB-INF**」ノードを展開し、Web.xmlファイルをダブルクリックします。
 - b. **Security**タブで「**Add**

Security Constraint」をクリックし、AuctionAppSOAPwsの制約を作成します。

- c. **Display Name**に**SSL transport for AuctionAppSOAPws**などの名前を入力します。
- d. Web Resource Collectionで「**Add**」をクリックし、図2のように**Resource Name**フィールドにSecure Areaと入力します。
- e. **URL Pattern(s)**フィールドに**/AuctionAppSOAPws/***と入力します。
- f. 「**Selected HTTP Methods**」オプションを選択し、「**GET**」と「**POST**」の各チェックボックスをオンにします。
- g. 「**OK**」をクリックします。
- h. 図3のように「**Enable Authentication Constraint**」オプションの選択が解除されていることを確認します。
- i. 「**Enable User Data Constraint**」オプションを選択していない場合は選択し、

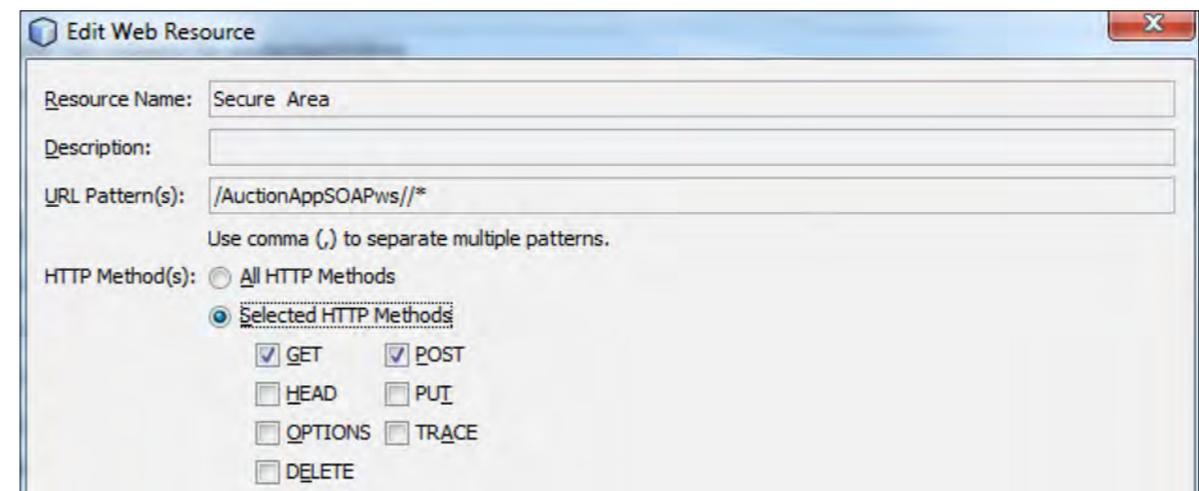


図2

Transport Guaranteeリストで「**CONFIDENTIAL**」が選択されていることを確認します。NetBeansにより、デプロイメント・ディスクリプタ内に適切な設定情報が生成されます。

- j. 「**Source**」タブをクリックして、すべての詳細を確認します。
- 3. Webサービス・アプリケーションのデプロイとテストを実行します。
 - a. 「**AuctionApp**」ノードを右クリックして、「**Deploy**」を選択します。
 - b. ブラウザを開き、リソースのURLである `https://localhost:8181/AuctionApp/AuctionAppSOAPws?wsdl` を入力します。図4のように、サーバーの証明書が表示されます。
 - c. 証明書を受け入れます。アプリケーションのWSDLファイルが表示されます。

Webクライアント

本項では、新しいWebサービス・クライアントの作成とセキュリティ保護を行い、前項でセキュリティを保護したWebサービスを参照するように設定します。そのために、クライアント・アプリケーションを作成します。

NetBeans IDE 7.2のWeb Service Clientウィザードを使用して、Webサービスのルックアップに必要なコードとすべてのファイルを生成します。

それでは、クライアント・アプリケーションを5分でコーディングします。

- 1. 初期状態のNetBeansプロジェクトを生成します。
 - a. Fileメニューで「**New Project**」

を選択します。

- b. Categoriesで「**Java Web**」を選択します。
- c. Projectsで「**Web Application**」を選択します。
- d. 「**Next**」をクリックします。
- e. プロジェクト名として `WebServiceClientSecureSSL` と入力して、「**Next**」をクリックします。
- f. サーバーがGlassFish Server (またはこれに類似した名称) となっていることを確認します。
- g. 「**Finish**」をクリックします。
- 2. Sellerエンティティを作成します。
 - a. `WebServiceClientSecureSSL` プロジェクトを右クリックし、「**New**」を選択して、「**Entity class**」を選択します。
 - b. **Class Name** フィールドに `Seller` と入力し、**Package** フィールドに `com.bonbhel.oracle.webServiceClientSecureSSL` と入力して、「**Next**」をクリックします。
 - c. 図5のように、Provider and Database画面で、Persistence Providerリストから「**EclipseLink (JPA 2.0) (default)**」を選択します。
 - d. Data Sourceリストで、「**jdbc/sample**」を選択します。jdbc/sampleはNetBeansに用意されているデータソースです。
 - e. 「**Finish**」をクリックします。
- 3. 手順2と同様の操作を実行して、ItemエンティティとBidエンティティを作成します。
- 4. Seller.javaファイルを開き、コード

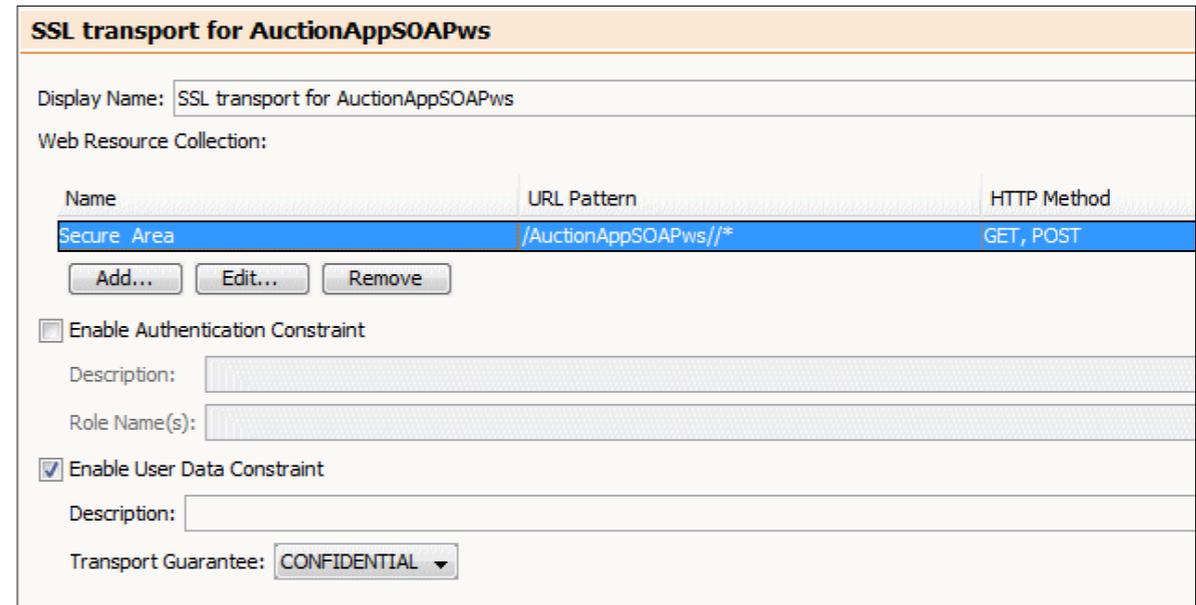


図3



図4

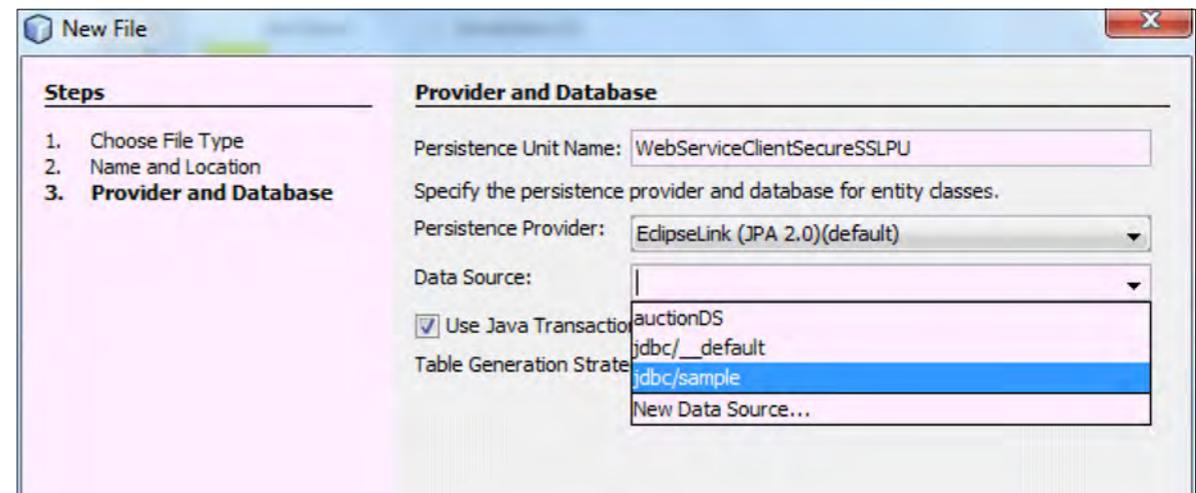


図5

内の任意の場所を右クリックして、「Insert code」を選択します。

5. 図6のようにGenerateウィザードで「Add property」を選択して、売主のプロパティ(String lastName、String firstName、String email)を追加します。
6. Item.javaファイルを開き、商品のプロパティ(String title、String description、Double initialPrice、Seller seller)を追加します。
7. エンティティ間の関係を定義するため、図7のようにNetBeansの警告を示す電球アイコンをクリックして、「Create bidirectional ManyToOne relationship」を選択します。この操作によって、Sellerエンティティにitemのリストが作成されます。
8. Bid.javaファイルを開き、商品のプロパティ(String bidderName、Double amount、Item item)を追加します。
9. エンティティ間の関係を定義するため、NetBeansの警告アイコンをクリックして、「Create bidirectional ManyToOne relationship」を選択します。
10. Sellerエンティティに作成されたitemのリストと、Itemエンティティに作成されたbidのリストのそれぞれについて、コード内の任意の場所を右クリックして「Insert code」を選択し、GetterとSetterを生成します。
11. 図8のように、Generateウィザードで「Getter and Setter」を選択します。この時点で、Seller.

javaファイルはリスト1のようになります。

12. JavaServer Faces(JSF)ページを作成します。作成するクライアントの実装は、先ほど作成したエンティティをベースとするJSFページで構成されます。
 - a. AuctionAppWebServiceClientプロジェクトを右クリックし、「New」を選択します。次に、Entity Classesで「JSF Pages」を選択し、「Add all」、「Next」の順にクリックします。
 - b. Session Bean Packageにcom.bonbhel.oracle.webServiceClientSecureSSL.facadeなどの名前を入力し、さらにJSF Classes Packageにもcom.bonbhel.oracle.webServiceClientSecureSSL.controllerなどの名前を入力します。
 - c. Folder NameにjsfClientなどの名前を入力します。
 - d. 「Finish」をクリックします。
13. Webサービス・クライアントを作成します。Web Service Clientウィザードを使用して、Webサービス・クライアントを作成します。ここで、JAX-WS Webサービスは、アプリケーション層に存在し、ネットワーク経由でアクセスされる外部サービスであると想定します。そのため、JAX-WS WebサービスのWSDLファイルに対するURLを使用します。
 - a. AuctionAppプロジェクトが稼働していることを確認します。稼働していない場合は、「AuctionApp」

ノードを右クリックし、「Deploy」を選択します。

- b. 「WebServiceClientSecureSSL」ノードを右クリックし、「New」を選択して、「WebServiceClient」を選択します。
 - c. 図9のように、New Web Service Clientウィザードで、ホストの完全修飾名を使用してWebサービスのWSDLファイルに対するURLを指定します(例:https://<ホストの完全修飾名>:8181/AuctionApp/AuctionAppSOAPws?wsdl)。
 - d. その他のすべてのデフォルト設定を受け入れます。パッケージ名はWSDLファイルから引き継がれます。
 - e. 「Finish」をクリックします。
14. クライアントのクラスに、JAX-WS Webサービスが提供するextrapolateAmountBidオペレーションを追加します。
 - a. WebServiceClient

```
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

図6

```
private Long id;
protected String title;
protected String description;
protected Double initialPrice;
protected Seller seller;

public void setId(Long id) {
    this.id = id;
}
```

図7

```
@ManyToOne
protected Seller seller;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

図8

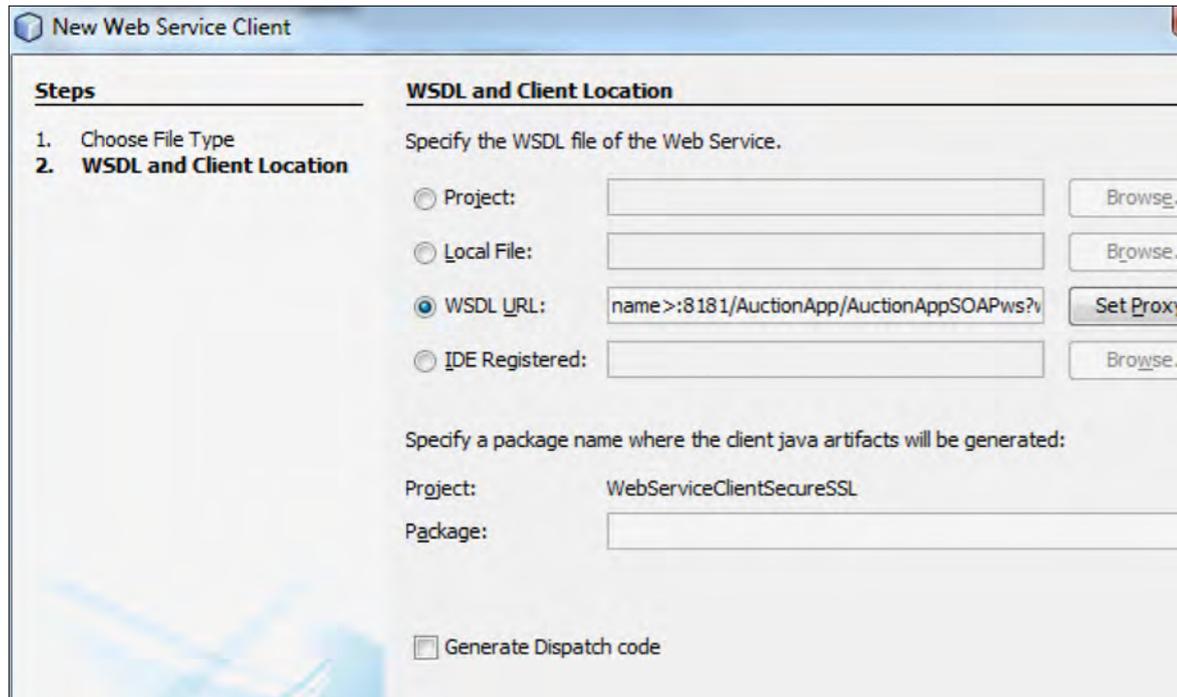


図9

SecureSSLプロジェクトの「**Source Packages**」ノードを展開し、コントローラのパッケージである `com.bonbhel.oracle.auction.AppWebServiceClient.controller` にある `BidController.java` ファイルをダブルクリックします。

- b. ソース・エディタ内の任意の場所にカーソルを置きます。
- c. `WebServiceClientSecureSSL` プロジェクトの「**Web Service References**」ノードを展開し、「`extrapolateAmountBid`」ノードをソース・エディタ内にドラッグします。「`BidController`」クラスのコードの最後に、**リスト2**のような `extrapolateAmountBid()`

メソッドが追加されます。**注:**別の方法として、ソース・エディタ内の任意の場所を右クリックし、「**Insert Code**」を選択することもできます。その場合、次に「**Call Web Service Operation**」を選択し、Select Operation to Invokeダイアログ・ボックスで「`extrapolateAmountBid`」オペレーションをクリックします。

- 15. ユーザーが入札エントリの編集や表示を行う場合に、入札額を(100倍として)予測するためには、`BidController` クラスにアプリケーション・ロジックを追加する必要があります。そのため、入札額を予測する `extrapolateAmountBid` オペレーションを呼び出します。
 - a. ソース・エディタで `BidController.`

LISTING 1 LISTING 2 LISTING 3

```
@Entity
public class Seller implements Serializable {
    @OneToMany(mappedBy = "seller")
    private List<Item> items;
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    protected String firstName;
    protected String lastName;
    protected String email;

    public List<Item> getItems() {
        return items;
    }

    public void setItems(List<Item> items) {
        this.items = items;
    }
}
```

[Download all listings in this issue as text](#)

javaファイルを開きます。

- b. `public String prepareView()` メソッドを**リスト3**のように編集します。

これにより、セキュリティ保護されたWebサービス・クライアントが再生成され、セキュリティ保護されたWebサービスのWSDLファイルを参照するようになります。

セキュリティ保護されたWebサービスのテスト

次に、このサービスをテストします。すなわち、セキュリティ保護されたWebサービスを、設定したクライアント・アプリ

ケーションから呼び出します。

本項では、前項で作成したセキュリティ保護されたWebクライアント・アプリケーション(`WebServiceClientSecureSSL`)とセキュリティ保護されていないWebクライアント(`AuctionAppWebServiceClient`)を使用して次のタスクを実行します。

- プロジェクトに含まれるセキュリティ保護されていないクライアント・アプリケーションからの、セキュリティ保護されたWebサービスの呼び出しの試行
- セキュリティ保護されたクライアントを使用した、セキュリティ保護されたWebサービスの呼び出し

1. セキュリティ保護されていないクライアントからセキュリティ保護されたWebサービスを呼び出して、予測入札額の表示を試行します。
 - a. AuctionAppプロジェクトが稼働していることを確認します。稼働していない場合は、「AuctionApp」ノードを右クリックし、「Deploy」を選択します。
 - b. AuctionAppWebServiceClientプロジェクトの「WebService」ノードを展開し、「AuctionAppSOAPws」ノードを右クリックして「Refresh」を選択します。
 - c. Confirm Client Refreshウィザードで、「Also replace local wsdl file with original wsdl located at:」を選択し、「Yes」



図10

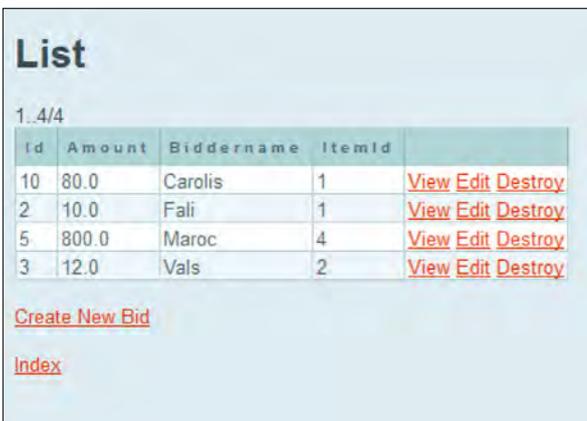


図11

2. セキュリティ保護されたWebクライアント(WebServiceClientSecureSSL)を使用してセキュリティ保護されたWebサービスを呼び出し、予測入札額を表示します。
 - a. AuctionAppが稼働していることを確認します。稼働していない場合は、「AuctionApp」ノードを右クリックし、「Deploy」を選択します。
 - b. 「WebServiceClientSecureSSL」プロジェクトを右クリックし、「Clean and Build」を選択します。
 - c. 「WebServiceClientSecureSSL」プロジェクトを再び右クリックし、「Run」を選択します。図10をクリックします。
 - d. 「AuctionAppWebServiceClient」プロジェクトを右クリックし、「Clean and Build」を選択します。
 - e. 「AuctionAppWebServiceClient」プロジェクトを再び右クリックし、「Run」を選択します。図10のように、すべてのエントリの一覧が表示されます。
 - f. 「Show all Bid Items」リンクをクリックすると、図11のように入札エントリの一覧が表示されます。
 - g. 入札者Faliに対する「View」リンクをクリックすると、図12のように、Faliの新たに予測された入札額が表示されます。図11と図12からわかるように、入札額が10.0から0.0に変わっています。これは、セキュリティ保護されたWebサービスの呼び出しにクライアントが失敗したことを示します。

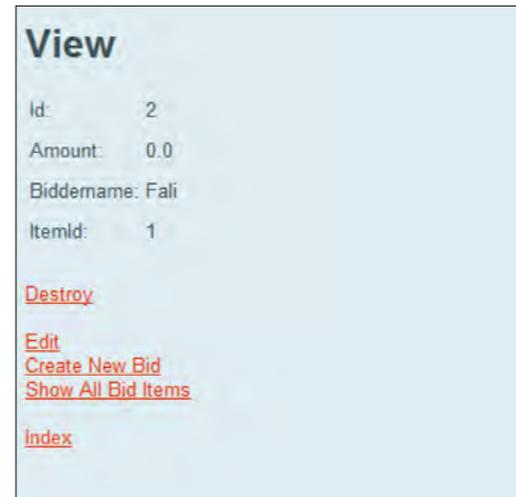


図12

のように、すべてのエントリの一覧が表示されます。

3. 「Show all Bid Items」リンクをクリックすると、図11のように入札エントリの一覧が表示されます。
4. 入札者Faliに対する「View」リンクをクリックすると、図13のように、Faliの新たに予測された入札額が表示されます。図11と図13からわかるように、入札額が10.0から1000.0に変わっています。これは、セキュリティ保護されたWebサービスの呼び出しにクライアントが成功したことを示します。

まとめ

本記事では、クライアント/サーバー間のデータ転送を保護するために、HTTPS経由でSSLを使用して既存のアプリケーションにトランスポート層のセキュリティを簡単に追加できることを確認しました。

NetBeansとGlassFishでは、これまでにないほど容易に、Webサービスにセキュリティを追加できます。

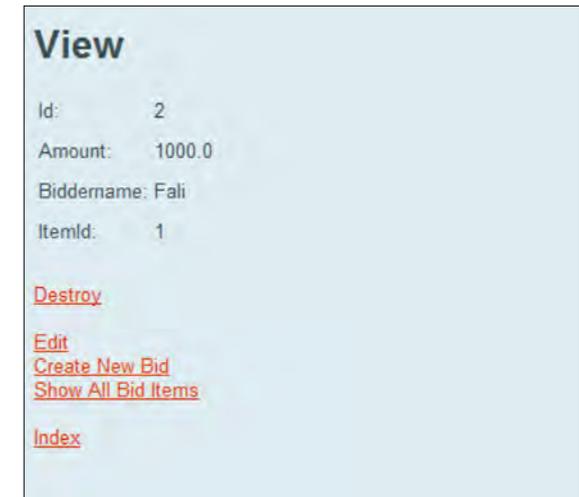


図13

第3回では、Webサービスの認証済みユーザーの作成に焦点を当てて説明します。 </article>

LEARN MORE

- [NetBeansの『高度なWebサービス相互運用性』マニュアル](#)
- [Metro User Guide](#)
- [GlassFishのリソース](#)

Project Lambdaの展望

Java SE 8で導入されるラムダ式の重要性についてJava言語アーキテクトのBrian Goetzに聞く JANICE J. HEISS

オラクルのJava言語アーキテクトBrian GoetzほどJavaプラットフォームを熟知している人はほとんどいないのではないのでしょうか。JavaチャンピオンのDick Wall氏は、Javaに関する新たな知見をどこから得ているかと尋ねられ、「Brian Goetzと話せばいつも面白い話が聞ける」と語っています。

Goetzは、ベスト・プラクティス、プラットフォームの内部仕様、並行処理プログラミングなどをテーマに80を超える記事を発表しています。また、2006年のJolt Awards最終選考に残り、2006 JavaOneカンファレンスで最多販売部数を記録した『Java Concurrency in Practice』の主著者でもあります。2006年8月にSun Microsystemsに入社する前はソフトウェア・コンサルタントとして、Javaテクノロジーに関する執筆活動だけでなく、カンファレンスでのたびたびの講演や、スレッド、Javaプログラム言語のメモリ・モデル、ガベージ・コレクション、Javaテクノロジーのパフォーマンスに関する通説などをテーマにプレゼンテーションも行ってきた経歴の持ち主です。

写真: BOB ADLER





「生産的な言語とは、多くの場合、明白な解決策と良い解決策とが一致する言語です」と語るJava言語アーキテクトのBrian Goetz

Java Magazine: Project Lambdaの導入により、「平均的な」Java開発者にはどのようなメリットがありますか。Project Lambdaのおかげでできる、逆に言えばProject Lambdaがなければできない、具体的なケースについて教えてください。

Goetz: Javaはすでにチューリング完全であるため、言語機能を追加してもプログラム・セットで表現できる幅は広がらないという考え方もあります。しかし、実践の場となると話はまったく別です。ある言語でどのような機能を利用できるかによって、その言語で簡単かつきれいに表現できるプログラムが決まります。そしてこの簡単かつきれいに表現できるという点こそが重要です。なぜなら、開発者が人間であるからです。生産的な言語とは、多くの場合、明白な解決策や簡単な解決策と良い解決策とが一致する言語です。人間はもともと怠惰なため明白で簡単な解決策を選んで失敗しがちですが、それが良い選択肢と一致していれば問題は起きませんから。

平均的な開発者の場合は、新しいコレク

ション操作APIの使用を通じて初めてラムダ式を使用するケースが多いでしょう。Project Lambdaは単なる言語機能ではなく、ライブラリも対象としています。言語機能とライブラリが一体となって、Javaプログラミング・モデルを大幅にアップグレードします。

ラムダ式(クロージャとも呼ばれます)がJava言語に追加されることで、開発者、特にAPIの設計者は、より洗練された方法で振る舞いを抽象化できるようになります。Javaにはすでに振る舞いを抽象化する方法がいくつかありますが、ラムダ式は簡潔に記述できるため、自然に書けるコードという観点で考え方を根本的に変えるものとなることは間違いありません。

コレクションについて考えてみましょう。コレクションに含まれる複数の要素を列挙するには、イテレータを用意して、そのイテレータから各要素を取得します。この操作は、次のようなfor-eachループで自動的に行われます。

```
for (Person p : people) {  
    // pを使用して処理を実行  
}
```

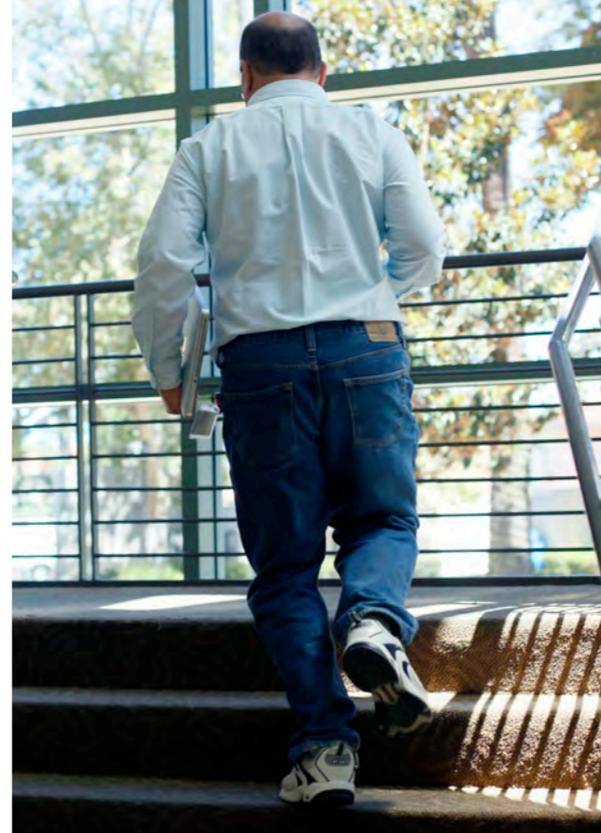
この記法は外部イテレータと呼ばれます。外部イテレータは非常に単純ですが、コレクションの各要素に対する処理とは関係のない付随的な部分も多く含まれています。外部イテレータは本質的に順次処理であり、要素をコレクションに格納されている順に処理しなければなりません。さらに、コレクションを利用する側がイテレータの構造を理解する必要があります。一方、内部イテレータを使用すると、各要素に適用する何らかの振る舞いを、利用側からコレクションに渡すことができます。たとえば、次のように記述します。

```
people.forEach(p -> { /* pを使用して  
処理を実行*/ });
```

特に違いはないように見えるかもしれませんが、ある非常に重大な変更が加えられています。それは、計算の構造がライブラリによって制御されているということです。利用側では処理の対象を指定する必要はありますが、処理の方法については指定していません。内部イテレータの良いところは、この「方法を知らなくてよい」という点にあります。たとえば、ライブラリで並列処理を使用する、あるいは複数の要素をより効率的な順序で処理するためにライブラリで把握しているメモリ内のデータの位置情報を利用するといったテクニックを駆使して、正しい答えをより早く得ることができます。(このようなテクニックを実行すべきかどうか、いつ実行するのか、どのように表すのかは依然として難しい問題です。しかし、少なくとも内部イテレータ・モデルでは、外部イテレータ・モデルでは決してできないことを実現できます。)ライブラリを開発する目的は、専門家が記述するコードを再利用できるようにすることです。そのため、「どのように処理を行うか」という方法の詳細をより多くライブラリに移すほど、そのライブラリの表現力と性能は向上します。

Java Magazine: ラムダ式の導入により、ライブラリの性質はどのように変わりますか。

Goetz: ラムダ式を導入すると、計算の制御はライブラリ側で行いながら、利用側でより簡単にカスタマイズできるAPIの開発が可能になります。そして結果的に、利用側では、より少ない手順でラムダ式を効果的に使用できるようになります。ラムダ式の導入によるこうした変化はJava言語開発者の作成するAPIに影響し、それによって現場で記述される



Java SE 8 のラムダ式の開発に奔走する Goetz

Javaコードの性質も影響を受けます。また、ツールボックスでラムダ式を利用することでAPIの「浸透性」が高まるという考え方もできます。外部イテレータでは、ライブラリの振る舞いと利用側とは厳密に区別されています。つまり、ライブラリはデータを一度に1要素ずつ渡し、利用側はただそのデータを受け取ります。一方、内部イテレータでは利用側とライブラリがより細かい粒度で振る舞いを分担し、それぞれにもっとも適した要素を提供し合います。

たとえば、「65歳以上の社員が1名以上いるグループを検索し、グループの人数の少ない順に出力する」という問題について考えてみましょう。現時点では次のようなコードを記述できます。

```
List<Employee> people = ...
Set<Group> groups = new HashSet<>();
```

```
for (Person p : people) {
    if (p.getAge() >= 65)
        groups.add(p.getGroup());
}
List< Group> sorted = new
ArrayList<>(groups);
Collections.sort(sorted, new
Comparator<Group>() {
    public int compare(Group a, Group b)
    {
        return Integer.compare(a.getSize(),
b.getSize());
    }
});
for (Group g : sorted)
    System.out.println(g.getName());
```

これは何とも見づらいコードです。groups やsortedといった、中間的な結果のみを保持するための使い捨てのような変数がいくつかあります。問題の内容に特化して書かれているため、問題を少し変えるだけで、コードには大幅な変更が必要になります。中間的な

コレクションヘデータを出し入れするために、何度もデータをコピーしています。そして、何を行っているかを把握するためには、コードを注意深く読まなければなりません。

Project Lambdaを導入したコレクション・ライブラリを利用すれば、この計算をもっと簡潔に表現できます。

```
people.filter(p -> p.getAge() > 65)
    .map(p -> p.getGroup())
    .removeDuplicates()
    .sorted(comparing(g -> g.getSize()))
    .forEach(g -> System.out.println(g.
getName()));
```

この例では余分な変数はなく、計算を上から下に追って読めば状況を正しく理解できます。このコレクション・ライブラリによって、ユーザーは細々とした関数の計算をパラメータ化できます。このパラメータ化という機能が、ユーザーの作業を簡単にする(多くの場合は効率的にもする)ようなライブラリ設計の原動力となっています。さらに、この例の場合、問題の内容を少し変えた程度では、コードに必要な変更は少なく済む可能性が高いのです。

Java Magazine: Project Lambdaの恩恵をもっとも受けるのはどのようなJava開発者ですか。

Goetz: 最終的にはだれもが恩恵を受けることとなります。ライブラリ設計者は、API設計のよりよい手段が得られますし、ユーザーはよりリッチなライブラリを利用できます。また、組織では性能に優れた強力なライブラリを利用しやすくなります。

Java Magazine: Project Lambdaを導入するためにもっとも大きな労力が必要になるのはどのような種類の開発者、またはどのよ





根本的な変革

ラムダ式は簡潔に記述できるため、自然に書けるコードという観点で考え方を根本的に変えるものとなることは間違いありません。

うな状況に置かれた開発者ですか。特定の条件下で、Project Lambdaが導入されることで開発者の仕事が難しくなる可能性はあるのでしょうか。

Goetz: 変化には必ずコストが伴います。Project Lambdaは単なる新しい構文の問題ではありません。いくつかの新しい概念が導入されているため、開発中のコードでProject Lambdaの新しい機能を利用する予定がない場合でも、開発者はJavaコードを読めるようになるためだけに、Project Lambdaを学ぶ必要があります。また、コアライブラリに対して大幅に機能が追加されるため、その追加機能についても学ぶ必要があります。しかし、Project Lambdaによる生産性と表現力の向上は、これらのコストを補って余りあるはずで。

Java Magazine: Project Lambdaと並列処理の関係について教えてください。

Goetz: これまでに挙げたいくつかの例は、より多くのことを表現できてエラーは起こりにくいようなコードを実現するライブラリの構築に関するものでした。この点は確かに重要な目標の1つです。しかし、もう1つ重要な目標があります。それは、ハードウェアの並列処理を活用するためのハードルを下げることで、Java SE 7では、多数のCPUコアで効率的に実行できるアルゴリズムを開発するため

のフォーク/ジョイン・フレームワークが追加されました。一方で、逐次アルゴリズムと並列アルゴリズムではソース・コードが大きく異なります。「65歳以上の社員がいるグループ」の問合せの並列実行をフォーク/ジョインを使用して記述したコードは、作成者以外にはなかなか理解できません。コーディングの量は1ページにも及び、同じ問題を順次的に実行するコードとはかけ離れたものになります。

並列処理もまたライブラリの中に移すことのできる機能の1つですが、このライブラリへの並列処理の移動は、外部イテレータ・モデルから内部イテレータ・モデルに変更することで可能になります。先ほどの問合せを少し変更すると、次のようにパラメータ化できます。

```
people.parallel()
    .filter(p -> p.getAge() > 65)
    .map(p -> p.getGroup())
    .removeDuplicates()
    .sorted(comparing(g -> g.getSize()))
    .sequential()
    .forEach(g -> System.out.println(g.
getName()));
```

変更したのは、`parallel()`の呼び出しと`sequential()`の呼び出しを挿入した箇所だけです。これで、`filter/map/sort`を並列実行できるようになります。最終的に複数の要素を順に処理したい場合には、逐次モードに戻します。この例からも明らかですが、ライブラリに並列実行を追加する作業が必要になるものの、内部イテレータ・モデルを使用することで、少なくとも並列実行が可能になります。

Java Magazine: Javaライブラリが古くなるという問題に対して、Project Lambdaはどのように対応しますか。

Goetz: Java 1.2でJava Collections

Frameworkが追加されて以来、約15年もの間、核となる抽象化の手法([Set](#)、[List](#)、[Map](#))は変更されていません。これは、インタフェースを一度公開すると、既存の実装クラスを壊すことなくインタフェースに新しい機能を追加することは不可能であるためです。皮肉にも、Java言語にラムダ式を追加することで、この問題はさらに深刻になります。ラムダ式を利用できるようになれば、すぐに[Collection.forEach](#)などの新しいメソッドを使用したくなるものです。(言語機能の追加は常に、既存のAPIに圧力をかけることになりま。たいていの場合、既存のAPIには、開発当初にその言語機能が存在したならば違う方法で記述されたであろうコードが含まれるからです。)

Javaにラムダ式を追加することで、もともと古くなっていたJava Collections APIがさらに古びて見えるという思いがけない結果となりました。そのため、Project Lambdaでは、ラムダ式を「デフォルト・メソッド」という別の機能と組み合わせることにしました。デフォルト・メソッド機能では、インタフェースに新しいメソッドを追加する際、そのデフォルトの実装(デフォルト・メソッド)を記述することで、互換性を維持できます。インタフェースを実装するクラスが該当するメソッドを実装していない場合には、デフォルト・メソッドが代わりに使用されるためです。デフォルト・メソッドはインタフェースの他のメソッドと同様、完全な仮想メソッドです。つまり、クラスではメソッドをオーバーライドしてもしなくてもかまいません。Project Lambdaではこのデフォルト・メソッドを利用して、既存のAPIに新しいメソッドを追加し、ラムダ式を活用しようとしています。

デフォルト・メソッドに類似する機能は他の言語にもありますが、Javaでのアプローチは少し異なります。C#には拡張メソッドの機

能がありますが、拡張メソッドは静的メソッドです。これに対して、Javaのデフォルト・メソッドは仮想メソッドです。C++では、多重継承が制限されていません。一方、デフォルト・メソッドでは振る舞いは多重継承できます(Javaではこれまでも型の多重継承は可能でしたが)、状態の多重継承はできません(状態の多重継承は、C++の継承に関する多くの問題の原因となっています)。Scalaにはトレイトがありますが、デフォルト・メソッドはトレイトよりも限定的です。これらの言語間の差は言語開発者の目的が異なるために生まれます。Javaでデフォルト・メソッドを追加する第一の目的は、インタフェースを進化させることでした。

Java Magazine: roject Lambdaを導入した新しいコレクションの例を見ると、「遅延」と「ストリーム」という2つの概念が浮かび上がります。遅延とは何で、ストリームとは何でしょうか。

Goetz: 遅延とは、処理の実行タイミングに関係する概念です。次のコードを見てみましょう。

```
bobs = people.filter(p ->
p.getFirstName().equals("Bob"));
```

このフィルタリングが実際に行われるのはいつでしょうか。選択肢は2つあります。[filter\(\)](#)メソッドから制御が戻るときにすべてのフィルタリングを終えるか、[bobs](#)から要素を引き出すときにフィルタリングを実行するかです。制御が戻るときに実行する場合は先行評価、[bobs](#)から要素を引き出す際に実行する場合

遅くても問題なし
遅延には、パフォーマンス上の大きな利点があります。
フィルタリングを先行して行う場合には、入力 of 全要素で条件を評価する必要があります。

は遅延評価と呼ばれます。先行計算か遅延計算かを選択できる点もまた、内部イテレータ・モデルを使用するライブラリで得られる実装の柔軟性の1つです。

遅延には、パフォーマンス上の大きな利点があります。フィルタリングを先行して行う場合には、入力の全要素で条件を評価する必要があります。しかし、1つ目の結果だけがほしいのに、リストに100万個もの要素が含まれているとどうなるでしょうか。この場合、多くの時間を無駄にしています(遅延では、無限のデータ・セットに対する計算も可能になります)。遅延を実現するためには、[filter\(\)](#)メソッドが、すべての要素

を含むコレクションではなく、ストリームを返す必要があります。ストリームはイテレータに似て、空になるまで値を引き出すことができますが、実際に次の要素を要求しない限り、フィルタリングが行われることはありません。計算処理をストリームとしてモデリングすることによって、フィルタリングやマッピングなどを行う際にデータを中間コレクションに詰める必要がなくなります。その代わりに、流れてくる値に対して処理を実行できます。そのため、前述のような問合せのコードが読みやすくなるばかりか、効率性も高まります。「65歳以上の社員のいるグループ」の順次問合せの例では、すぐに破棄することになる中間コレクションに要素を設定する必要がなくなります。また、並列化した場合は、フィルタリング、マッピング、並べ替えを行う場合に、それぞれ個別のパスを用意することなく、1本の並列パス内で実行できます。遅延はこれらの特徴を実現できる機能です。



Java Magazine: ラムダ式やJava SE 8の現在の開発段階において、Javaコミュニティからどのようなフィードバックを受けたいですか。どのような質問であれば答えてみたいと思われますか。また、コミュニティはどう

すれば開発段階のコードをチェックできますか。

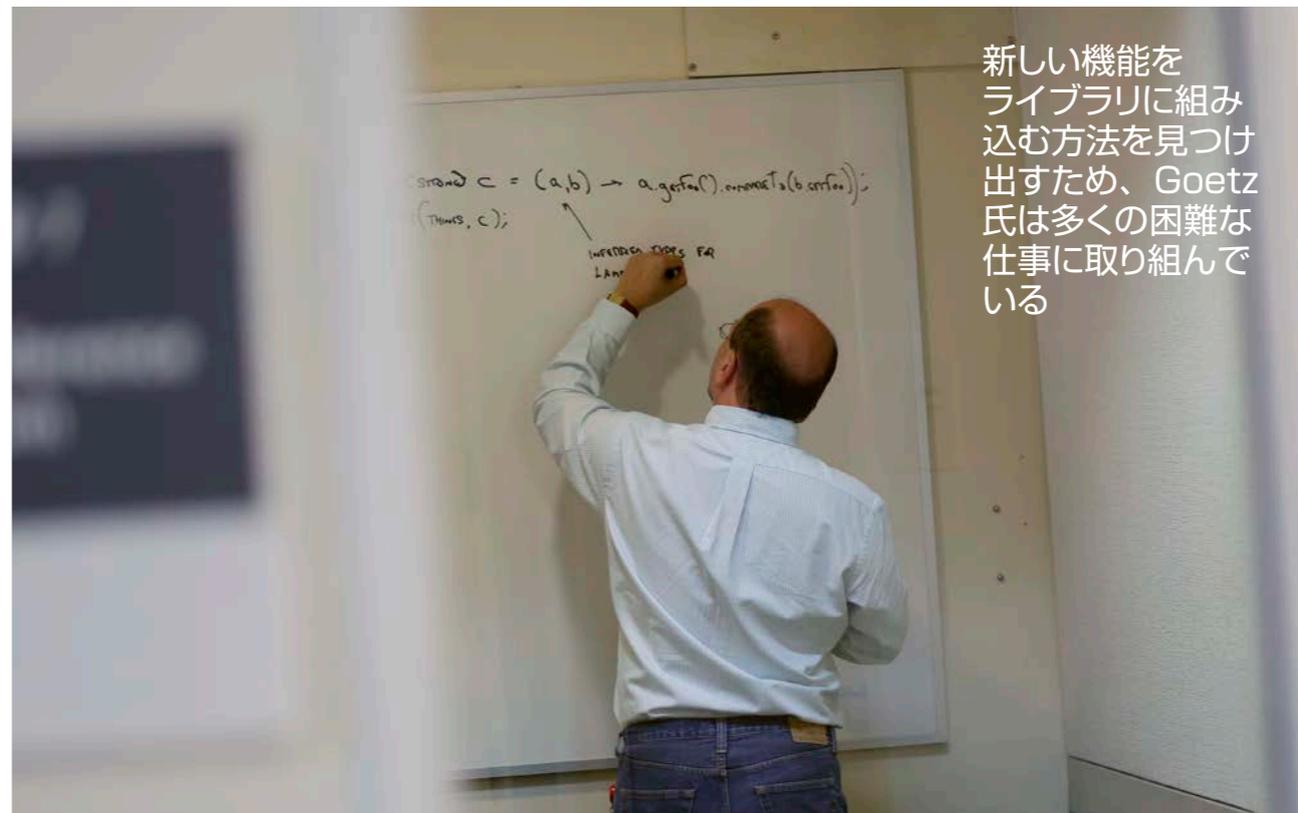
Goetz: コミュニティがProject Lambdaにできるサポートとして、もっとも重要で唯一の方法は、試していただくことです。バイナリをダウンロードし、新しいCollectionsの機能やラムダ式をご自身のコードで実際に試して、その結果をご報告ください。

Java Magazine: 一部の開発者は、ラムダ式は非常に優秀ではあるものの、作成者以外がコードを理解し保守していくことは非常に難しいのではないかと心配しているようです。このような心配に対してご意見をいただけますか。

Goetz: 記述されたコードの理解と保守はリスクではありますが、これはラムダ式がなくても存在するリスクです。非常に明解できれいな読

みやすいコードをJavaで記述することは可能です。そして、わかりにくく整理不足で混乱を招くようなコードを記述することもまた可能なのです。几帳面なプログラマーはラムダ式を活用して、より明解できれいな読みやすいコードを記述できますが、無頓着なプログラマーはおそらく、これまで以上にわかりにくく、整理されておらず、混乱を招くようなコードを記述してしまうでしょう。

JDK 8で予定されている変更の隠れたメリットとして、可変性が穏やかに抑制されることがあります。前述の例では、「古い」バージョ



新しい機能をライブラリに組み込む方法を見つけ出すため、Goetz氏は多くの困難な仕事に取り組んでいる

ンではあらゆる変更ができました。たとえば、中間結果のコレクションに要素を設定したりコレクション自体を変更したりできました。しかし、2つ目のバージョンでは、利用側から変更できないため、結果的に間違いが起きにくくなります。ハードウェアではマルチコア・システムの導入が進んでいますが、こうした動向にとっても、可変性の抑制は適しています。

Java Magazine: What open questions in Lambda still have to be dealt with?

Goetz: Project Lambdaで対処する必要のある既知の問題はどのようなものですか。Goetz:ライブラリに関連する作業がまだ多く残っています。今回、いくつかの例を使用して機能を説明しましたが、例で示した機能をライブラリに実際にどう組み込むかは検討中の段階です。 </article>

Janice J. Heiss: オラクルのJava編集者であり、Java Magazineのテクノロジー編集者

LEARN MORE

- [Project Lambda](#)
- [Project Lambdaバイナリのダウンロード](#)
- [JSR335: Lambda Expressions for the Java Programming Language \(Javaプログラミング言語のラムダ式\)](#)
- [Brian Goetzのブログ](#)

慎重に対応する

几帳面なプログラ

マーは、ラムダ式を使用してより明確できれいで読みやすいコードを記述できますが、**無頓着なプログラマー**はおそらく、さらにわかりにくく整理不足で混乱を招くようなコードを記述してしまうでしょう。



BEN EVANS AND
PETER LAWREY

BIO

パート2

Java HotSpot VMの内部を探る (2) : パフォーマンス解析のための統計情報

マクロ・レベルのベンチマーク、分布形状の理解、そしてクライアント視点による計測が重要な理由

本シリーズの第1回では、Java HotSpot VMについて紹介し、メソッドの実行時コンパイルに関連する基本的な概念の一部を説明しました。第1回をまだお読みでない方は、本記事の前にぜひ一読ください。

本記事では、Javaプラットフォーム上のパフォーマンス解析で特に注意を要する点をいくつか説明します。具体的には、メソッド計測結果の統計情報の扱い方、特にそれがマイクロベンチマークの場合について見ていきます。

インターネット上にはJavaのパフォーマンスについて誤った結論を導き出しているブログや記事が散見されます。誤りの理由は、Javaのマイクロレベルのパフォーマンス計測値がもつ統計上の性質を考慮していないことにあります。

本記事を執筆することで、マイクロベンチマークの正確さに潜む問題を明らかにし、小さなメソッドやコードのごく一部ではなく、マク

ロレベル(アプリケーションおよびアプリケーション層の全体のレベル)でのベンチマークを重視するよう読者の皆さんを後押しできれば幸いです。

まずは前回の記事で使用したコードの修正版を見ていきます(**リスト1**参照)。

前回使用したコードは、Just-In-Time(JIT)コンパイルおよび単純なメソッド(getter/setterなど)のインライン化の効果を確認するためのものでした。本記事で扱うコードも基本的には同じですが、さらに次の2つの機能を追加しています。

- 繰り返しごとに少量のオブジェクト・アロケーションを実行する機能(最終的にガベージ・コレクションが発生)
 - 繰り返しの1回ごとにかかった実行時間を収集できる計測機能
- 前回の記事では、平均値を計算しただけで、それぞれの実行が実際に発生したタイミングは考慮していませんでした。しかし、本記事

で調査する統計的側面を十分に検証するためには、より高い精度の情報が必要になります。そのため、**リスト2**のクラスを使用して、それぞれの実行を管理します。

2010年モデルのMacBook Pro上でJava SE 7u4を使用して、オブジェクトのアロケーションを行わずに実行した場合の出力を見てみます(**リスト3**参照)。

実際の実行時間の計算された平均値は44.9ナノ秒ですが、実行時間の50パーセンタイル値は0です。この0という値は、コードの実行にまったく時間がかからなかったことを示します。平均値とパーセンタイル値は矛盾しているように思われます。

実は、この矛盾の原因は、Mac OS Xの計時サブシステムにあります。Mac OS Xの計時サブシステムでは時刻の記録が

マイクロ秒レベルでしか行われないため、`System.nanoTime()`が返す値は1,000ナノ秒(=1マイクロ秒)単位になるのです。この結果から、特定のプラットフォームで生成されるパフォーマンス計測値に影響を及ぼしうるハードウェアやOSの機能に注意することが非常に重要であるとわかります。

Mac OS X以外のOSやハードウェアでは、`nanoTime()`の精度はナノ秒です。これ以降は、ハードウェアとJava仮想マシン(JVM)は引き続き同じものを使用しますが、OSをUbuntu Linuxに変更します。これにより、それぞれの実行について、本記事で必要としている細かい精度の数値を取得できます。

**非正規な分布
JVMから取得したパフォーマンス測定値は、通常は正規分布に従いません。**

結果の分布
パフォーマンス・チューニングに適した数値を収集するた

BEN EVANS氏の写真:
JOHN BLYTHE



めには、パフォーマンスの影響を受ける結果の分布を理解することが非常に重要です。特に、JVMから取得されるパフォーマンス測定値は、通常は正規分布に従いません。

メソッドの実行時間などのパフォーマンス測定値は、通常、基本的な結果とJVMによるノイズという2つの部分で構成されます。一般的に、JVMの引き起こすノイズの量は常に正数となり、場合によっては計測中の値よりもはるかに大きくなります。

そのため、分布の形状は正規分布(ガウス分布または釣鐘型分布とも呼ばれます)ではなく、非対称なファット・テール(裾の厚い)分布に近い形状となります。全体的には図1のようなガンマ分布に類似した形状になりますが、値の大きな外れ値(統計において他の値から大きく外れた値)が存在します。

外れ値がどのように発生するかを確認するために、JVMのもっとも重要な構成

要素の1つであるガベージ・コレクション(GC)の影響について検討します。周知のとおり、JVMのマークアンドスイープ・アルゴリズムでは、すべてのアプリケーション・スレッドが定期的に一時停止状態となります。したがって、計測対象のメソッドの実行中にGCが開始されることもありえます。その結果、メソッドの実行速度は、通常よりも大幅に遅く見えることとなります。計測対象のメソッドがGCの実行中に一時停止状態となるためです。このように、GCは、計測結果に深刻な影響を及ぼす場合があります。

GCの影響を実際に確かめるため、2回の実行結果(1回はオブジェクトのアロケーションなし、もう1回はオブジェクトのアロケーションあり)で、外れ値の問題がどの程度現れるかについて調べることになります。この問題を詳細に解説するために、特定のパーセンタイル・レベルに達するまでにかかった時間を出力します。出力結果を読み取る際には、「99%」の

ところに出力された実行時間は、全測定結果の99パーセントがこれを下回るような値である、ということに注意してください。

リスト4は、2010年モデルのMacBook Pro、Ubuntu Linux 11.04、JDK 7u4の組み合わせでオブジェクトのアロケーションなしで実行した場合の結果であり、リスト5は、同じ組み合わせでオブジェクトのアロケーションあり

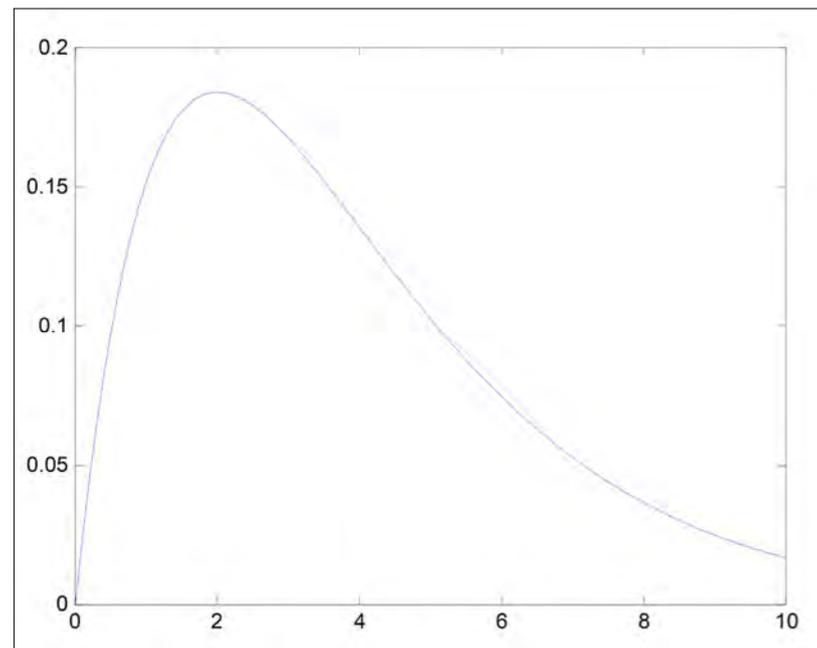


Figure 1

LISTING 1 LISTING 2 LISTING 3

```
import java.util.UUID;
import java.util.concurrent.Callable;

public class GetSetCallerWithAlloc implements
Callable<Double> {
    private final int runs;
    private final long[] results;
    private final boolean doAlloc;

    public GetSetCallerWithAlloc(long[] res, boolean alloc) {
        results = res;
        runs = res.length;
        doAlloc = alloc;
    }

    @Override
    public Double call() {
        ViaGetSet getSet = new ViaGetSet();
        double sum = 0;
        UUID uuid;
        long end, start = System.nanoTime();
        for (int i = 0; i < runs; i++) {
            getSet.setOne(getSet.getOne() + 1);
            sum += getSet.getOne();
            if (doAlloc) {
                uuid = UUID.randomUUID();
                sum += uuid.toString().length();
            }
            end = System.nanoTime();
            results[i] = end - start;
            start = end;
        }
        return sum;
    }
}
```

Download all listings in this issue as text

で実行した場合の結果です。

リスト4、リスト5の結果から言えることがいくつかあります。1つは、オブジェクトのアロケーションを行わない場合でも、外れ値となる結果がいくつか存在することです。これらの外れ値は、OSのスケジューラの動作などの影響を受けて発生しています。

ただし、もっとも大きな影響は、オブジェクトのアロケーションを行った場合に見られます。オブジェクトのアロケーションを行った場合の出力では、90パーセント値が23ナノ秒から5.00マイクロ秒に増加しており、2桁の差があります。オブジェクトのアロケーションを行うことで、計測しようとしている結果がまったく隠されてしまっています。これが、JVMでのマイクロベンチマークのコーディングに潜む危険性です。マイクロベンチマークは、JVMを低レベルで扱うことに精通した専門家のみが実行すべき非常に特殊なケースと言えます。

ほとんどの開発者は、コードのごく一

部に対してベンチマークを行ってそこから広範囲の結果を推定するのではなく、より大きな計測対象、すなわち全体のスループットやエンド・ツー・エンドの応答時間など、アプリケーションのビジネス目標に直接関連するような計測対象に的を絞るべきです。

統計的な厳密さ

Javaのマイクロパフォーマンスの計測結果には前述のような分布の性質があるため、標準的な統計手法を使用する場合に注意する必要もあります。特に、標準偏差などの手法は、このような性質の結果を扱う場合に非常に信頼性の低いものになりかねません。

標準偏差などの手法は、利用時に考慮すべき注意点を考慮せずに使用されることがあまりにも多くあります。多くの人は、「値の68パーセントは平均値±標準偏差の範囲に存在し、95パーセントは2倍の標準偏差の範囲に存在する」という「規則」は覚えていても、その規則が、分布が正規分布に従っているという前提に立っていることは忘れていたのです。

正規分布に従っていない場合は、任意の確率分布のパーセンタイルに関する法則は存在しません。

さらに、統計情報を万全な状態で取り扱うためには、計測結果が正規分布に従っていないということ以外にも考慮が必要な点があります。特に、独立した実行を十分な回数行うことで、外部のノイズ（他のプロセス、OSのタスク、その他のシステム的な影響）によって結果が大きく歪められた回の影響を受けないようにする必要があります。

前述の結果は、独立した実行を十分な回数繰り返す中から選択された結果であり、全体的な統計情報を正しく表していることが検証されています。

JVMから取得した統計結果の適切な取り扱い方の基準となる論文に、『Statistically Rigorous Java Performance Evaluation』（Georges, Buytaert, Eeckhout、

LISTING 4

LISTING 5

```
71 1,000,000
110 1,000,000
50.0% delay was 23 ns
90.0% delay was 30 ns
99.0% delay was 43 ns
99.9% delay was 164 ns
99.99% delay was 248 ns
99.999% delay was 3,458 ns
99.9999% delay was 17,463 ns
```

```
field, getter/setter=33.1 ns
field, getter/setter=25.1 ns
```

 [Download all listings in this issue as text](#)

2007年)があります。この論文は複数のWebサイトで入手できます。非常に高いレベルの厳密性と正確性を必要とするアプリケーションでは、この論文で説明されている手法に厳密に従うことをお勧めします。

最後に、ここまで触れてこなかった計測の注意点の中で非常に重要なものを紹介します。

クライアントの視点

Webサーバーなどのシステムについて、応答時間を計測する場合を考えてみます。これまで説明した計測手法によると、一部のサーバーのページ応答時間が長くなりそうです。ここで長い応答時間を示すページは、GCによる一時停止の開始時に処理中であった不運なページです。

しかし実際には、事態はさらに複雑です。サーバーの視点から処理時間を考えるだけでは十分ではありません。クライアントからのリクエストは、JVMがGCのために一時停止している間にも送信され

続けるからです。こういったリクエストの処理はGCサイクルの完了後に開始されるので、処理時間の計測だけでは、リクエストの処理がGCによって遅延したという兆候は示されません。

しかし、リクエストを送信するクライアントの視点からは、リクエストは確実にGCの影響を受けており、実行速度が遅く感じられます。

上記の理由で、応答時間などのパフォーマンス計測値を本当に正確に理解するためには、結果の分布の性質に加えて、クライアントの視点でとらえた全体の処理時間についても考慮する必要があります。

まとめ

本記事では、JVMのマイクロベンチマークの性質について説明し、良好なマイクロベンチマークの開発に潜む次のような危険性について確認しました。

- JVMの動的で自動管理されているという性質がパフォーマンス計測値に影響すること

注意点

標準偏差などの手法は、手法を利用する場合に適用すべき注意点を考慮せずに使用されることがあまりにも多くあります。

- GCは特に大きな影響を及ぼす可能性があること
- マイクロベンチマークの計測結果は正規分布に従わないこと
- 単純な記述統計(平均や標準偏差)はマイクロベンチマークでは誤解を生じやすいこと

ほとんどのアプリケーション開発者は、アプリケーションやコンポーネントの全体を対象とし、より大きな尺度でベンチマークを行うことを目指すべきです。さらに、大きな尺度でベンチマークを行う場合でも、開発者が従うべき基本的なベンチマークの原則は次のように数多くあります。

- パフォーマンス・インジケータの分布の全体的な形状を理解することが重要であり、統計手法を利用する際には、はじめに分布の全体的な形状を確認する必要があること
- 独立した実行を十分な回数行って、統計の安定性を確立すること
- 「ネットワークのキューイング」の影響に気をつけること。十分に正確な計測を行うためには、クライアントの視点から計測する必要があること [</article>](#)

LEARN MORE

• [Java HotSpot VM](#)



MAKE THE FUTURE
JAVA



FIND YOUR JUG HERE

I have met so many amazing people through the London Java Community— friends, mentors, new colleagues— and through it I have achieved much more than I could have on my own.

Trisha Gee
London Java Community (LJC)

[LEARN MORE](#)





JULIEN PONGE



NIO.2の新ファイル・システムAPIの紹介

Java SE 7でリリースされた新ファイル・システムAPIを活用するreleased in Java SE 7

Java SE 7では、[JSR 203](#)のいわゆるNIO.2 APIと呼ばれる機能がNIOパッケージに新たに追加されました。この新機能の大部分は、ファイル・システムAPIと非同期チャンネルを対象とするものです。

従来、Javaではファイル・システム操作のほとんどを、システムに依存する機能を抽象化した[java.io](#)パッケージのクラスを使用して実行していました。しかし、[java.io](#)パッケージのクラスには、ファイル権限の操作やシンボリック・リンクの処理などに制約がありました。これらの制約を克服するために多くのライブラリが開発されましたが、そのようなライブラリには、ネイティブ・ライブラリまたは外部プロセスの呼

び出しに頼らざるをえないという代償があります。また、ファイル・システムの一般的な操作(コピー、移動、探索など)を容易に実行するためのライブラリや、仮想ファイル・システム(圧縮アーカイブ、FTP、HTTPなど)を構築するためのライブラリなども開発されてきました。

本記事では、ファイル・システムに関連するNIO.2の追加機能、具体的にはファイル/ディレクトリ操作、探索、メタデータ操作、変更の監視に焦点を合わせて説明します。これから詳しく見ていくとおり、NIO.2 APIによって、Javaツールセットに優れた機能が追加されます。

最初のステップ

NIO.2の追加機能の大部分は[java.nio.](#)

[file](#)パッケージに含まれています。まずはNIO.2 APIの重要なインタフェースである[Path](#)を紹介しましょう。

パスとファイル: [Path](#)インタフェースは、ある特定のファイル・システム内にあるファイルまたはディレクトリを抽象化します。[Path](#)のインスタンスを取得するためには[Paths](#)クラスを使用します。[Paths](#)クラスには2つの[get\(\)](#)メソッドがあり、それぞれURIと可変数のパス要素を引数に取ります。たとえば、**リスト1**のように記述した場合は、`/etc/hosts`ファイルへの[Path](#)の参照を取得できます。

[Path](#)の参照には、2つのパスを比較するメソッドや、他のパスからの相対パスを解決するメソッドなどが含まれています。一方、次のような一般的なパス操作を実行するためには、通常は[Files](#)クラスを使用します。

- ファイル、ディレクトリ、シンボリック・リンクの作成

- コピー、移動、削除
- 属性の問合せ
- ファイル・システム・ツリーに対する反復処理
- 読取り/書込み用のストリームおよびチャンネルの取得
- 直接的な読取り/書込み操作の実行

[Path](#)と[Files](#)のメソッドをすべて紹介するだけでは面白みに欠けるので、本記事では代わりにいくつかの例を見ていきます。

単純な操作: まず、一時ディレクトリを作成し、ファイルをそのディレクトリ内にコピーします。

リスト2では、一時ディレクトリ(例: `/var/folders/cj/h0x5ghcd1tz5s7shj8txxf00000gn/T/nio21324767717145250358`)内で、asmファイルをasm-copy.jarファイルとしてコピーしています。

[CopyOption](#)パラメータは複数指定することも、まったく指定しないこともできます。もっとも頻

繁に使用されるCopyOptionパラメータはStandardCopyOption列挙型の定数です。**リスト2**では、所有者、グループ、権限などのファイル・システム属性を維持するように指定しています。この他に、アトミックな移動操作の実行を示す定数と既存ファイルの置き換えを示す定数も使用できます。また、コピー操作でシンボリック・リンクを辿らないようにするためのLinkOption列挙型も使用できます。

シンボリック・リンクについて一言触れておくと、たとえば**リスト3**のように、元のASMライブラリのJavaアーカイブ(JAR)に対するシンボリック・リンクが存在しない場合に、シンボリック・リンクを1つ作成してJARのサイズを問い合わせることができる。

支払うべき代償

java.io パッケージのクラスの制約を克服するために**多くのライブラリが開発**されましたが、そのようなライブラリには、ネイティブ・ライブラリまたは外部プロセスの呼び出しに頼らざるをえないという代償があります。

高速の読み書き

従来、ファイルの内容の読取りと書込みを行うためには、多くのコードを記述する必要がありました。しかし、NIO.2のFilesクラスには、ファイルの内容の読取りと書込みを1回の操作で実行できる、高速のヘルパー・メソッドが複数用意されています。これらのヘルパー・メソッドは、文字

列またはRAWバイト配列を扱います。

リスト4は、ファイルの内容を読み取る方法を示しています。また、**リスト5**のように、ファイルに新しい内容を書き込む操作も簡単に実装できます。

既存のストリームやNIO APIとの統合: NIO.2 APIは単体での使用を目的としてはいないため、既存のストリームおよびNIO APIと組み合わせて使用する手段が用意されています。

java.ioスタイルのI/Oストリームとバッファは、FilesクラスのメソッドnewInputStream()、newOutputStream()、newBufferedReader()、newBufferedWriter()を使用して取得できます。また、NIOチャネルはnewByteChannel()メソッドを使用して取得できます。

興味深いことに、コピー先としてjava.io.OutputStreamのインスタンスを使用できるcopy()メソッドもあります。

使用例としては、次のようにファイルの内容を標準出力ストリームにダンプする操作が挙げられます。

```
Files.copy(hello, System.out);
```

ファイル・システムの操作

FileSystemクラスは、その名のとおり、ファイル・システムを抽象化します。ファイル、ディレクトリ、ユーザー、グループへのアクセスやファイル・システムの変更通知など、多くの便利なサービスを提供しています。

ファイル・システムへの参照の取得と利用: FileSystemは抽象クラスです。インスタンスは、対応するFileSystemsファクトリ・オブジェクトを使用して取

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6

```
// These two calls are equivalent
Path hosts1 = Paths.get("/etc/hosts");
Path hosts2 = Paths.get("/etc", "hosts");
// (...)
```

 [Download all listings in this issue as text](#)

得できます。

FileSystemには、デフォルトのファイル・システム(Java仮想マシンが起動されたファイル・システム)を返すgetDefault()メソッドがあります。デフォルトのファイル・システムへの参照は、次のようなコードで取得できます。

```
FileSystem fs = FileSystems.
getDefault();
```

ファイル・システムのプロパティの一部は、ファイル・システムの種類によって異なります。たとえばファイル・システムのルートはUNIX系のオペレーティング・システムでは1つですが(/)、Microsoft Windowsではパーティションおよびドライブに1つずつ存在します(C:\、D:\など)。ある特定のファイル・システムに存在するルートのは、getRootDirectories()メソッドで確認できます。getRootDirectories()メソッドはIterable<Path>を返します(**リスト6**参照)。

FileSystemのインスタンスを使用して、Pathインスタンスを取得することもできます。Pathには、2つのパスの比較、サブパスの抽出、2つのパス間におけ

る相対パスの取得などを行う操作メソッドがあります。

FileSystemオブジェクトからPathのインスタンスを取得するためには、getPath()メソッドを呼び出します。getPath()メソッドは可変数の文字列を引数に取ります。**リスト7**のように、各文字列はパスの一部を表します。

2つのパスの比較は次のようにして実行できます。

```
assert main.compareTo(alsoMain)
== 0;
```

compareTo()は、多くの類似メソッドと同様に、「小なり(<)」を表す-1、「等しい(=)」を表す0、または「大なり(>)」を表す1の値を返します。

リスト8では、パスをURI表現またはファイル表現に変換しています。また、**リスト9**のように、パスに対する操作を実行することもできます。

ファイル・システム・プロバイダのリスト: FileSystemsには他にも、ファイル・システム内のURIまたはパスに対応するFileSystemインスタンスを作成するためのファクトリ・メソッドがあります。これらのファクトリ・メソッド



は、zipアーカイブ、HTTP接続やFTP接続、あるいはSMBネットワーク・ドライブ上で、具象プロバイダが仮想ファイル・システムとして振る舞うような場合に特に便利です。このファクトリ・メソッドの機能を利用して、「現実の」ファイル・システムと仮想ファイル・システムを同じように処理できます。

`FileSystemProvider`クラスを使用することで、利用可能なファイル・システム・プロバイダを問い合わせることができます。`FileSystemProvider`クラスは、`java.nio.file.spi`パッケージに含まれるサービス・プロバイダ・インタフェース(SPI)です。

リスト10のコードは、Java SE 7のほとんどのインストール環境において、ファイルとJAR URIスキームの両方を返します。仕様上はJARスキーム・プロバイダを取得できるようにする必要はないのですが、JDK 7とOpenJDKではJARスキーム・プロバイダの情報を提供しています。このSPIの機能を利用して、独自のファイル・システム・プロバイダ(HTTP、FTPなど)を作成することも可能ですが、この説明については本記事では割愛します。

これ以降は、あるzipファイルをファイル・システム・プロバイダとして開いて「探索」する操作を通じて、NIO.2 APIをさらに深く考察します。

ファイル・システム・プロバイダの探索
`java.io.File`クラスには`list()`メソッドと`listFiles()`メソッドがあります。これらの

メソッドは`File`インスタンスがディレクトリを表す場合に使用でき、ディレクトリ直下にある子ファイルおよび子ディレクトリのリストを返します。

一方、`java.nio.file.Files`クラスには`java.io.File`クラスよりもやや強力なメカニズムが備わっています。探索要素の選択的なフィルタリングとマッチングを実現する、ファイル・ビジターとディレクトリ・ストリームです。

本項では、このファイル・ビジターとディレクトリ・ストリームの機能について説明します。

zipファイル・システムを開く: JARをファイル・システムとして開きます (**リスト11**参照)。

`FileSystems`クラスには、リスト11に示した以外の引数を取る`newFileSystem`ファクトリ・メソッドもあります。たとえば、設定マップをパラメータとして受け取り、プロバイダ固有のキーを使用するメソッドがあります。**リスト11**では、`Path`と`ClassLoader`を引数に取っています。これはファイル内のファイル・システムを開く場合に適したメソッドです。

`FileSystem`は`AutoCloseable`を実装しています。そのため、`try-with-resources`文の中で使用した場合、例外がスローされたかどうかにかかわらず`FileSystem.close()`メソッドが確実に呼び出されます。

パス・マッチング: ここでは、アーカイブ内にあるすべての.classファイルを出力する場合を考えてみます。

傑出した機能
NIO.2 APIによって、Javaツールセットに優れた機能が追加されます。

LISTING 7 / LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
Path main    = fs.getPath("src", "discover", "nio2", "fs", "Main.java");
Path alsoMain = fs.getPath("src/discover/nio2", "fs", "Main.java");
Path src     = fs.getPath("src");
```

 [Download all listings in this issue as text](#)

`PathMatcher`を利用することにより、`getPathMatcher()`メソッドで、`Path`オブジェクトを正規表現またはファイル・グlobs構文とマッチングできます (**リスト12**参照)。

パス・マッチングの表現は、「glob:<グlobs表現>」または「regexp:<正規表現>」という形式に従います。たとえば、「glob:/foo/*.java,class」という表現は、/fooディレクトリ内部のすべての.javaファイルおよび.classファイルと一致します。

ファイル・ビジター: Visitorデザイン・パターンは、ファイル・システム・ツリーを探索し、ファイルに対して何らかの操作を行う必要がある場合に適しています。`FileVisitor<T>`インタフェースには、次の状況に対応するメソッドが定義されています。

- ディレクトリへのアクセスの前後に通知されるビジター
- ファイルへのアクセス時に通知されるビジター
- ファイルへのアクセス中に例外が発生した場合に通知されるビジター

また、ファイル・ビジターのメソッドでは、`FileVisitResult`列挙型の定数を返します。この定数は、探索の続行、探索の終了、兄弟(同レベルにあるファイルまたはディレクトリ)のスキップ、サブツリーのスキップのいずれかを指示します。さらに、オーバーライド可能な`SimpleFileVisitor<T>`というデフォルト実装もあります。たとえば**リスト13**のように、アーカイブ・ファイル・システムをルートから探索し、.classファイルのパスを出力できます。

現時点では`visitFile()`メソッドの`attrs`パラメータは無視してかまいません。ファイル属性の操作方法については後ほど説明します。

ディレクトリ・ストリーム: ファイル・ビジターはカスタマイズ性に優れていますが、1つのディレクトリ内で特定のファイルを検索するだけの場合は、よりシンプルなソリューションであるディレクトリ・ストリームを利用できます。ストリームとは一言で言えば、特定のフィルタに一致するパスのセットに対して実行される反復可能(Iterable)な

オブジェクトです。また、ディレクトリ・ストリームはクローズする必要があるため、DirectoryStreamインタフェースはAutoCloseableインタフェースも継承しています。そのため、try-with-resources文の中で使用できます。

リスト14では、前述のJARのルートにあるすべてのエントリを検索しています。また、**リスト15**では、グロブ構文によるフィルタを渡してJavaクラス・ファイルのみをリストしています。

ストリームのフィルタ条件としてDirectoryStream.Filterオブジェクトを渡すことで、より複雑な問合せを実行することも可能です。たとえば、前述のグロブ・マッチング・オブジェクトを再利用して、すべてのJavaクラスをリストできます(**リスト16**)。

ファイル・ビジターを使用する場合は、PathMatcherの問合せに基づいたフィルタ条件を指定する必要はありません。たとえば、「サイズが1024キロ・オクテットを超えるすべての実行ファイル」など、ファイル・サイズやファイル属性を指定した問合せを作成することもできます。

ファイル・ツリー全体のzipアーカイブへのコピー: ファイル・ツリー探索、Filesクラスの操作、一時的なファイル・システム作成の機能を組み合わせて、現在のディレクトリ・ファイル・ツリー全体をUTF-8でエンコードしたzipアーカイブに簡単にコピーできます(**リスト17**参照)。

ファイルをzipアーカイブ

にコピーするのではなくFTPファイル・システムを使用してリモートのFTPサーバーにコピーする場合も、**リスト17**のコードの大部分をそのまま使用できます。変更する必要があるのはファイル・システムの作成部分のみです。Files.createDirectories()メソッドでは、作成するディレクトリに至る必要なディレクトリをすべて作成できます。

メタデータの操作

NIO.2 APIでもっとも興味深い機能の1つが、ファイル・システム・レベルでメタデータを操作できる機能です。以前は、ファイルの所有者、グループ、アクセス権限などの情報を標準のJavaランタイムから取得するためには、外部プロセスの実行やネイティブ・コードとの統合を行う必要がありました。しかし、これにより外部プロセスに固有の問題やネイティブ・コードに固有の問題が新たに発生しました。メタデータの情報を取得することは可能になったものの、多種多様なオペレーティング・システムやファイル・システムが存在するため、ランタイム環境が提供する機能に合ったコードを作成する必要があります。

java.nio.file.attributeパッケージで、そのようなメタデータへのアクセスがサポートされています。通常は、ファイル属性ビューの階層構造を利用します。各ファイル属性ビューは、基盤となるファイル・システムの機能に応じて操作可能なメタ

注目の機能

NIO.2 APIでもっとも興味深い機能の1つが、ファイル・システム・レベルでメタデータを操作できる機能です。

LISTING 13 LISTING 14 LISTING 15 LISTING 16 LISTING 17 LISTING 18

```
Files.walkFileTree(fs.getPath("/"), new
SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes
attrs)
        throws IOException {
        if (matcher.matches(file)) {
            System.out.println(file);
        }
        return FileVisitResult.CONTINUE;
    }
});
```

 [Download all listings in this issue as text](#)

データ一式を表します。

BasicFileAttributeViewインタフェースには、既存のファイル・システムに共通する属性(ファイル・サイズ、

最終更新日、作成日など)が定義されています。DosFileAttributeViewのインスタンスでは、DOS系のファイル・システムの属性(ファイルが隠し



ファイル、システム・ファイル、アーカイブかどうか)が定義されます。同様に、`PosixFileAttributeView`にはPOSIXシステムの属性が定義されており、`AclFileAttributeView`ではアクセス制御リスト(ACL)をサポートするファイルシステム上のACLにアクセスできます。

ファイル属性ビューの詳細な説明は、`java.nio.file.attribute`パッケージのJavadocに記載されています。ここからは、一般的な使用パターンについて確認します。

ファイル属性:特に一般的なファイル属性には、`Files`クラスのメソッドから直接アクセスできます。`size()`、`isDirectory()`、`isRegularFile()`、`isSymbolicLink()`、`isHidden()`、`getLastModifiedTime()`などのメソッドがあります。

1つの属性へのアクセス、あるいは1つの属性の変更には、`getAttribute()`メソッドと`setAttribute()`メソッドを使用できます。`getAttribute()`と`setAttribute()`は、ファイル属性ビューの識別子をプリフィックスとして付加した、属性を表す文字列を引数に取ります。各ファイル属性ビュー・インタフェースの引数の文字列と属性の戻り型は、該当する

Javadocに定義されています。

リスト18のように、ファイルのPOSIX権限を問い合わせることができます。出力結果は**リスト19**のようになります。さらに、

`BasicFileAttributes`の任意のサブクラスに対して`readAttributes()`メソッドを使用して、固有の属性にアクセスできます(**リスト20**)。

注目すべき点として、`Files.readAttributes(Path, BasicFileAttributes, LinkOption[])`では読取りの一括処理を実行します。この一括処理は、複数の属性を同時に読み取る場合にパフォーマンス面で有効です。

POSIXファイル権限は、**リスト21**のようにして直接アクセスできます。より一般的なケースについては、最初にファイル属性ビューにアクセスすることで対処できます。たとえば、あるファイルのPOSIXファイル所有者を取得するためには、**リスト22**のコードを使用できます。

POSIXファイル権限は通常、`rw-r-xr-x`などの文字列形式で表現します。`rw-r-xr-x`は、ファイルの所有者には読取り権限、書き込み権限、実行権限があり、グループ・メンバーとその他のユーザーには読取り権限と実行権限のみがあることを意味します。このようなPOSIX権限の表示と変更は**リスト23**のコードで可能です。出力結果は次のようになります。

```
Old permissions: rw-r--r--
New permissions: rwxr-xr-x
```

FileStoreクラス:すべてのパスに、対応するファイルの保存先(ボリューム、パーティションなど)を基本的に表すファイル・ストア・オブジェクトが関連付けられています。`FileStore`クラスは、ファイル・ストレージで特定のファイル属性ビューがサポートされているかどうかを問い合わせる際に便利です。また、

LISTING 19

```
Posix permissions for asm-all-2.2.3.jar
-> OTHERS_READ
-> OWNER_READ
-> GROUP_READ
-> OWNER_WRITE
```

 [Download all listings in this issue as text](#)

ディスク使用状況の統計情報にもアクセスできます。

リスト24のコードでは、Mac OS Xで次のような出力を生成するいくつかの問合せを実行しています。

```
/dev/disk0s2
hfs
169030116 / 244277768
true
true
false
```

変更の監視

一部のアプリケーションでは、ファイルシステムの変更を監視する必要があります。たとえば、アプリケーション・サーバーには、アプリケーション・アーカイブが特定のフォルダに置かれたことを検知して

自動的にデプロイするものがあります。同様に、統合開発環境では、サード・パーティ・アプリケーションによるプロジェクト・ファイルの追加、削除、変更を監視する場合があります。他にも、複数のアプリケーションがファイルの保管場所でデータを交換するようなアプリケーション統合パターンの例もあります。

ファイル・システムの変更の監視を簡単に実装する方法は、あるフォルダのエントリを定期的に確認し、各ファイルの最終更新タイムスタンプに基づいて追加、削除、または変更されたファイルを判別することです。しかし、たとえばLinuxの`inotify`のように、より効率的なオペレーティング・システム・レベルの監視メカニズムも存在します。

NIO.2には、オペレーティング・システムに利用可能な監視機能がある場合は

追加機能へのアクセス
NIO.2の追加機能の大部分はjava.nio.fileパッケージに含まれています。

それを利用し、ない場合は定期的なポーリング処理に戻すという便利なAPIが用意されています。[WatchService](#)は、特定のPathオブジェクトの変更について通知できるサービスです。監視対象とするイベントのタイプ、すなわち追加、変更、削除、またはオーバーフロー(通知システムですべてのイベントをレポートできないことを示すイベント)を指定できます。たとえば、**リスト25**のコードは、/tmp内の変更を監視し、イベントを出力します。

[WatchService](#)は、監視するパスのファイル・システムから取得します。[WatchService](#)もまた[AutoCloseable](#)を実装したオブジェクトであり、最終的に確実にクローズされるようにtry-with-resources文の中で使用することをお勧めします。さらに[WatchService](#)をパスに登録すれば、イベントをキューから取得できるようになります。

[WatchKey](#)は登録と組み合わせて使用され、登録時に返される、監視キーを表すオブジェクトです。[WatchKey](#)は、イベントの監視時にも取得できます。そのためには、[WatchService](#)の[poll\(\)](#)メソッドまたは[take\(\)](#)メソッドを使用します。リスト25の例では、新しいイベントを取得できるまで待機する余裕があるため、[take\(\)](#)を使用することにしました。[WatchKey](#)では状態が維持されるため、このキーの有

効性を判定する問合せを行った上で、最新の監視イベントを取得することが可能です。

キーの使用後、再利用する場合はリセットしておく必要があります。[reset\(\)](#)メソッドはBoolean値を返します。この戻り値は、キーが無効になった場合(監視対象フォルダが消失した場合など)にfalseとなります。また、[cancel\(\)](#)メソッドを呼び出して、監視サービスの登録を取り消すこともできます。

最後に、キーにはイベントに関する情報が含まれます。たとえば、イベントの種類に関する情報(通常は[StandardWatchEventKinds](#)を使用)や、Pathのインスタンスであるコンテキスト・オブジェクトを保持できます。コンテキスト・オブジェクトは、サービスで監視していたイベントの種類([ENTRY_CREATE](#)、[ENTRY_MODIFY](#)、[ENTRY_DELETE](#))に対応します。

まとめ

本記事では、Java SE 7のいわゆるNIO.2パッケージの一部としてリリースされた新しいファイル・システムAPIについて紹介しました。NIO.2パッケージを使用した場合、一般的なファイル操作と高度なファイル操作の両方を簡単に実行できます。高度なファイル操作の例としては、フォルダ内の変更を効率的に監

フィルタリングとマッチング

java.nio.file.Filesクラスにはjava.io.Fileクラスよりも強力なメカニズムが備わっています。探索要素の選択的なフィルタリングとマッチングを実現する、ファイル・ビジターとディレクトリ・ストリームです。

LISTING 25

```
import static java.nio.file.StandardWatchEventKinds.*;

// (...)

Path tmp = Paths.get("/tmp");
try (WatchService watcher = tmp.getFileSystem().newWatchService()) {
    boolean running = true;
    tmp.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
    while (running) {
        WatchKey key = watcher.take();
        for (WatchEvent<?> event : key.pollEvents()) {
            Path path = (Path) event.context();
            WatchEvent.Kind<?> kind = event.kind();
            System.out.println(kind + " => " + path);
        }
        running = key.reset();
    }
}
```

 [Download all listings in this issue as text](#)

視する操作や、メタデータを維持しながらファイルを圧縮zipアーカイブに移動する操作などがあります。また、重要なこととして、NIO.2パッケージは既存のjava.ioクラスやjava.nioクラスを容易に統合でき、さらに新しいタイプのファイル・システム・プロバイダをすべて同じ方法で操作するように拡張できます。本記事で紹介した例以外にも、JDK 7ディストリビューションにNIO.2コード・サンプルが含まれていますので(sample/nio/file)、ぜひご確認ください。 </article>

LEARN MORE

- [Java SE 7 API](#)
- [「\[ファイルI/O\(NIO.2を含む\)\] Javaチュートリアル](#)
- [OpenJDKのNIOプロジェクト](#)
- [「\[Better Resource Management with Java SE 7:Beyond Syntactic Sugar\]](#)



ADAM BIEN



Java EE Connector Architecture 1.6

コンテナ・サービスとの無駄がなく深い統合を実現するJava EE Connector Architecture 1.6

依存性注入(DI)またはJNDIルックアップを利用することで、永続化やメッセージングなどの主要なリソースに直接アクセスできます。非標準のリソースを統合してアプリケーションにインジェクションすることも簡単ですが、プーリング、監視、セキュリティ、粒度の細かいトランザクションなどのコンテナ・サービスを利用できない可能性があります。

各リソース・タイプに対応するコンテナ・サービスを実装する作業は手間がかかります。Java EE Connector Architecture (JCA)では、そのようなコンテナ・サービスを容易に利用できる標準的なメカニズムが提供されます。

本記事では、JCAに基づくキー/値ストア形式のファイル・リソース・アダプタを実装する方法を説明します。この実装では、ファイル数を最小限に抑え、XMLは使用せず、アノテーションのみを使用して設定を行います。

JCA 1.6—期待以上の使いやすさ

JCA 1.6の目的は次のとおりです。

“Java EE Connector Architectureにより、Java EEプラットフォームを各種EISに接続する標準アーキテクチャが定義されます。EISの例としては、エンタープライズ・リソース・プランニング(ERP)システム、メインフレームのトランザクション処理(TP)システム、データベース・システムが挙げられます。Java EE Connector Architectureでは、各種EISをアプリケーション・サーバーおよびエンタープライズ・アプリケーションと統合するための、スケーラビリティに優れたセキュアなトランザクション・メカニズムを定義しています”(JCA 1.6仕様、JSR 322)

JCAは元々、複数の大規模エンタープライズ・システムを統合することを目的として策定されましたが、あらゆる外部システムへのアクセスに使用できるシンプルな仕様となっています。少数のクラスを作成するだけで、機能を網羅したリソース・アダプタ(コネクタ)

を実装できます。JCA仕様のバージョン1.6ではXMLデプロイメント・ディスクリプタと同様の設定を行うアノテーションが導入されているため、XMLデプロイメント・ディスクリプタの利用が必須ではありません。これにより、開発者の作業が楽になります。ワーク管理、セキュリティ、トランザクションの状態などのコンテナ・サービスに直接アクセスできるため、コネクタの実装を利用する方が、アプリケーション側でこれらのサービスを実装するよりも簡単です。

本記事では、**リスト1**のAPIを使用してローカル・ファイル・システムにアクセスするアウトバウンド・コネクタを実装します。

リスト1で定義したAPIインタフェースは、アプリケーション・コード内に間接的にインジェクションされます。JDBCと同様に、JCAのアウトバウンド通信のアーティファクトはコネ

クション型であるため、アプリケーション側でコネクションを取得し、このコネクションを(たとえば`connection.close()`を呼び出して)プールに戻す必要があります。本記事の例のBucketインスタンスは、**リスト2**のBucketStoreをコンポーネントBean(EJB Beanまたはサーブレット)内でインジェクションすることで作成できます。

このBucketとBucketStoreのファイルを専用のJARパッケージベースのMavenプロジェクト内に配置し、アプリケーションから参照できるようにします(**リスト3**参照)。

BucketStore APIをコンパイルするためには、JCA APIに含まれる`javax.resource.Referenceable`インタフェースが必要です。`Referenceable`はGlassFish 3.1.2

組み込みライブラリとしてJava EE 6 APIに付属しています。BucketStoreファイルの親プロ

容易な利用法

JCA 1.6では、コンテナ・サービスを容易に利用できる**標準的なメカニズム**が提供されます。



ジェクト内で、このライブラリを1回参照しています。また、この親プロジェクトは、すべてのモジュールの整合性を保ってビルドするためにも使用します(リスト4参照)。

一般的に、コネクタ実装のほとんどは、APIモジュールと実装モジュールに分けられます。本記事のBucketインタフェースはAPIであり、jarパッケージタイプでビルドされます。このBucket APIモジュールは、プロジェクト・オブジェクト・モデル(POM)によってjarパッケージタイプでビルドされますが、さらにリソース・アダプタ・アーカイブ(RAR)プラグインを使用して拡張されます(リスト5a、リスト5b参照)。

アプリケーションとリソース・アダプタ実装の両方が、コンパイル時に上記のAPIクラスに依存します。この共通するJava EE 6 APIへの依存関係のみを親プロジェクト内で宣言することで、両方のリソース・アダプタ・モジュールの整合性を保ってビルドすることが容易になります。クライアント・モジュールはオプションのサブモジュールであり、テストやデモの目的で使用できます。

必要な構成

一般的に、リソース・アダプタ(コネクタ)の実装は、外部システムとアプリケーション・サーバーとの双方向の関係を確立できます。アプリケーション・サーバーはリソース・アダプタに、トランザクションの状態、コネクションのライフサイクル、セキュリティについて通知します。

一方、リソース・アダプタは最小限のメタデータ・セットをアプリケーション・サーバーに公開することが期待されます。

JCA 1.6より、XMLデプロイメント・ディスクリプタ(ra.xml)の代わりにアノテーションでメタデータを指定できるようになりました。本記事の例のFileAdapterクラスには、コンポーネント定義のためのアノテーション(@Connector)が付与されています。@Connectorはおもに、このアノテーションを付与されたコンポーネントが(.rar形式でパッケージ化されることとは別に)リソース・アダプタであることを示します(リスト6a、リスト6b参照)。

本記事のFileAdapterクラスではBootstrapContextを参照する必要はなく、その他のコールバック呼び出しにも関心がないため、すべてのメソッドを空のままにしています。

本質から取りかかる

リスト7aとリスト7bのとおり、BucketStoreの実装は難しいものではありません。

FileBucketStoreはConnectionFactoryとして振る舞うため、BucketStoreの実装を通じてReferenceableインタフェースを実装する必要があります。ReferenceableインタフェースとSerializableインタフェースは共に、このAPIで記述されているJNDI空間への登録を適切に実行するために使用されます。

Referenceableインタフェースの実装に必要な作業は、javax.naming.Referenceインスタンスを管理するための2つのアクセサ・メソッドを実装することだけです。より興味深いのが、ファクトリ・メソッドのgetBucketです。例では、getBucketを呼び出

LISTING 1 LISTING 2 LISTING 3 LISTING 4 LISTING 5a LISTING 5b

```
public interface Bucket extends AutoCloseable{

    void write(String file,byte[] content);
    void delete(String file);
    byte[] fetch(String file);
    @Override
    void close();

}
```

 [Download all listings in this issue as text](#)

すたびに新しいバケットを作成するのではなく、ConnectionManagerにManagedConnectionFactoryを渡し、これを使用してコネクションを割り当てるよう依頼しています

allocateConnectionメソッドの2つ目のパラメータに、ConnectionRequestInfoの匿名インスタンスを渡しています。匿名であることから、すべてのコネクションが同一と見なされます。そして、equalsメソッドとhashCodeメソッドの両方で、ConnectionRequestInfoのすべて

のインスタンスが同一であるという結果を返すように実装を行っています。ただし、このコネクタの将来のバージョンで、現在の実装を拡張して認証を導入することも考えられます。匿名のConnectionRequestInfoの代わりに「null」を渡すこともできますが、その場合、移植性に欠けたコードとなる可能性があります。

ローカル・ファイル・システムとのトランザクションのやり取りはすべて、FileBucketクラス内に実装しています。FileBucketでは、コネクタのAPIの

Bucketインタフェースを実装するだけでなく、AutoCloseableインタフェースのcloseメソッドも実装しています。AutoCloseableインタフェースの実装によって、アプリケーションでtry-with-resourcesのメカニズムを使用して、不要なコネクションをクローズできます(リスト8a~リスト8e参照)。

ただし、FileBucketクラスでは単に、close()のすべての呼び出しを、コンストラクタで渡されたCloseableインタフェースのインスタンスに即座に委譲しています。FileBucketではbegin、commit、rollbackの各コールバックメソッドによって、トランザクションの進行状況を把握します。

アプリケーションはこのFileBucket実装を使用して、ローカル・ファイル・システム内のコンテンツにアクセスすることになります。FileBucket実装では書き込み操作や削除操作を直接実行せず、キャッシュしています。トランザクションのコミット時に、flushChangesメソッドを呼び出し、キャッシュの内容をローカル・ファイル・システムに書き込みます。

一方、rollbackが呼び出された場合はキャッシュをクリアするため、永続ストア(ローカル・ファイル・システム)は影響を受けません。また、トランザクションの開始時に、createIfNotExistメソッドを呼び出しています。このメソッドは、作業ディレクトリ内にルート・フォルダを作成します。

再利用可能な部分

FileBucketStoreはFileBucketクラスのファクトリです。FileBucketStoreクラスとFileBucketクラスはいずれも

ファイル・アクセス機能を実現するクラスであるため、その実装はドメインに大きく特化しています。FileBucketではJCA APIにまったく依存せずにファイル・アクセス機能を実装しているため、単体テストを容易に実行できます(リスト9a、リスト9b参照)。

残る2つの必要なクラスが、javax.resource.spiパッケージのManagedConnectionインタフェースの実装クラスとManagedConnectionFactoryインタフェースの実装クラスです。GenericManagedConnectionはManagedConnectionインタフェースの実装クラスであり、FileBucketインスタンスを効果的に管理する物理的なコネクションを表します。GenericManagedConnectionは名前のおり、ドメインに特化した実装に対する依存度が低く、さまざまなリソース・アダプタ(コネクタ)実装で再利用できます(リスト10a~リスト10f参照)。

GenericManagedConnectionのほとんどのメソッドについては、説明の必要はないでしょう。重要な機能はfireConnectionEventメソッドに実装されています。fireConnectionEventメソッド内で配信されるイベントにより、ManagedConnectionの状態がアプリケーション・サーバーに通知されます。このコールバック・メカニズムによって、アプリケーション・サーバーではコネクション・プールの適切な管理とクリーンアップを行うことができます。イベント通知のメカニズムが存在しない場合、このマネージド・コネクション・プールは枯渇し、アプリケーションの実行がブロックされてし

LISTING 6a / LISTING 6b / LISTING 7a / LISTING 7b

```
import javax.resource.ResourceException;
import javax.resource.spi.ActivationSpec;
import javax.resource.spi.BootstrapContext;
import javax.resource.spi.Connector;
import javax.resource.spi.ResourceAdapter;
import javax.resource.spi.ResourceAdapterInternalException;
import javax.resource.spi.TransactionSupport;
import javax.resource.spi.endpoint.MessageEndpointFactory;
import javax.transaction.xa.XAResource;
```

```
@Connector(reauthenticationSupport = false,
transactionSupport =
TransactionSupport.TransactionSupportLevel.Local-
Transaction)
```

 [Download all listings in this issue as text](#)

まいます。本記事の例のコネクタではローカル・トランザクションのみがサポートされるため、getXAResourceメソッドではResourceExceptionをスローして、拡張アーキテクチャ(XA)トランザ

クションがサポートされないことを示す必要があります。

GenericManagedConnectionFactoryの実装はさらに少し難しい内容になっています(リスト11a、リスト

11b、リスト11c参照)。

`GenericManagedConnection`クラスでは、細かな処理だけではなく、アウトバウンド通信のためのアーティファクトを表す`@ConnectionDefinition`アノテーションなどの重要なメタデータを提供しています。アプリケーション・サーバーでは`@ConnectionDefinition`アノテーションに指定された情報を使用して、ドメインに特化したコネクタの実装が検索されます。また、`setRootDirectory`メソッドに付与されているアノテーションも興味深い内容です。`@ConfigProperty`アノテーションは、リソース・アダプタのユーザーが設定できるプロパティを示すものであり、使いやすいユーザー・インタフェースを生成するために使用します。Bean Validation仕様(JSR 303)で定義されている`@Min`アノテーションは、値の設定前にユーザー入力値を検証するために使用します。

`matchManagedConnection`メソッドでは、既知のすべてのコネクションから`GenericManagedConnection`インスタンスを識別して抽出する必要があります。`ConnectionRequestInfo`パラメータは、該当する`GenericManagedConnection`インスタンスを識別する目的で使用します。

リソース・アダプタを持つキー/値ストア
`FilesResource`はコネクタの機能を、EJB Beanとして実装したRESTfulサービスとして公開します。`BucketStore`は`@Resource`アノテーションを使用して、任意のJava EEコンポーネントに簡単にインジェクションで

きます(リスト12a、リスト12b参照)。

インジェクションされた`BucketStore`インスタンスは`Bucket`ファクトリとして振る舞います。`FilesResource`の例では、`Bucket`インスタンスを`try-with-resources`文の中で宣言しています。`Bucket`インスタンスは`java.lang.AutoCloseable`を実装しているため、`try`文が正常終了したか異常終了したかに関係なくクローズされます。

`BucketStore`は、`@Resource`アノテーションに有効なコネクタ・リソースのJNDI名を指定することでインジェクションできます。インジェクションに必要なJNDI名は、コネクタのデプロイ中に設定されます。

コネクタのデプロイとインストール

スタンドアロン・コネクタはRARファイルとしてパッケージ化されます。このRARファイルは、最初にデプロイする必要があります。GlassFishでは、次のコマンドをコマンドライン・インタフェース(CLI)で実行するか、またはWebベースの管理コンソールを使用して、このRARをデプロイできます。

```
$GLASSFISH_HOME/bin/asadmin
-port 4848 deploy --force ./
store/target/jca-file-store.rar
```

コネクタのデプロイが正常に終了しても、アプリケーション・サーバーとクライアントとの統合に必要な設定を完了するまでは、コネクタは利用できません。この設定には、ホスト名、ポート、ユーザー、パスワードなどの適切な外部エンドポイント情報が必要となる場合があります。

LISTING 8a LISTING 8b LISTING 8c LISTING 8d LISTING 8e

```
import java.io.*;
import java.nio.file.Files;
import java.nio.file.LinkOption;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Map.Entry;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentSkipListSet;
import javax.resource.ResourceException;
import org.connectorz.files.Bucket;

public class FileBucket implements Bucket {

    private String rootDirectory;
    private ConcurrentHashMap<String, byte[]> txCache;
    private Set<String> deletedFiles;
    private Closeable closeable;
    private PrintWriter out;

    public FileBucket(PrintWriter out, String rootDirectory,
        Closeable closeable) {
        this.out = out;
        this.rootDirectory = rootDirectory;
        this.closeable = closeable;
        this.txCache = new ConcurrentHashMap<>();
        this.deletedFiles = new ConcurrentSkipListSet<>();
    }
}
```

 [Download all listings in this issue as text](#)

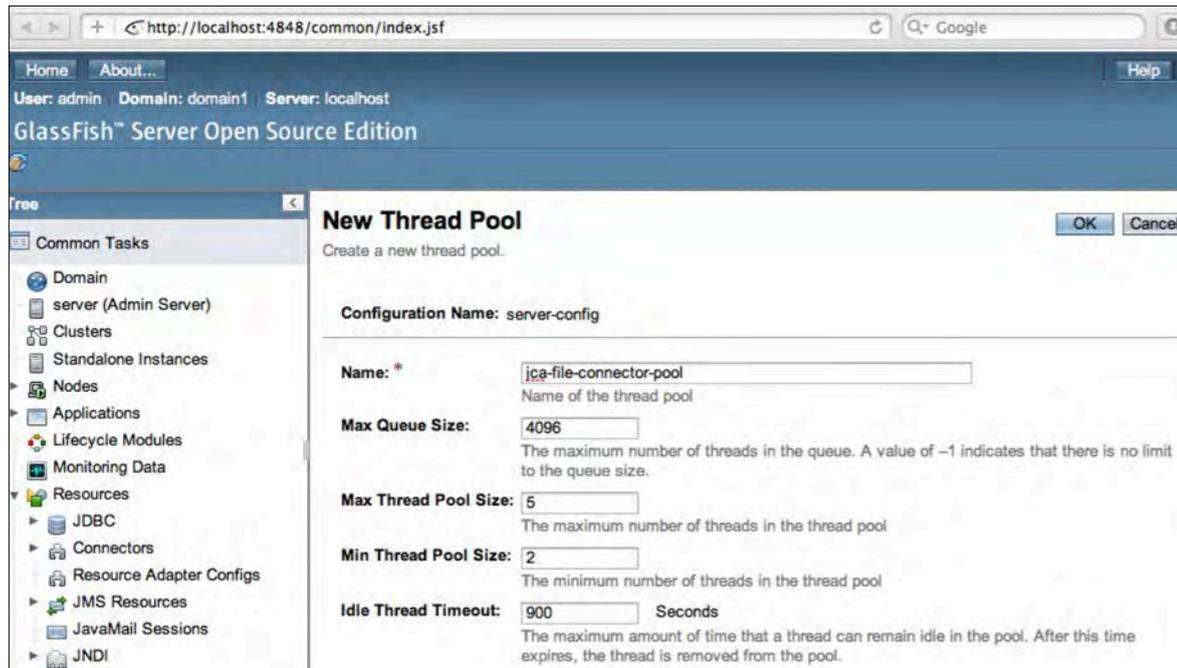


図 1

手順1:スレッド・プールを設定します。専用のスレッド・プールを使用することで、管理と監視が大幅に容易になります。GlassFish GUIのNew Thread Poolウィザード(図1参照)で、新しいスレッド・プールの作成と設定を行うことができます。

本記事の例のファイル・コネクタでは新しいスレッドを開始するわけではなく、**WorkManager**の機能も利用しません。ファイル・システムへのアクセスには**WorkManager**を使用しないため、ここでは**WorkManager**の設定は不要です。ただし、非同期のファイル・コネクタを実装する場合は、通常は専用のスレッド・プールを利用することになります。

手順2:リソース・アダプタを設定します。デフォルト値を使用してプールを作成した後は、New Resource Adapter Configウィザード(図2参照)で、新しいリソース・アダプタの設定を作成し、先ほ

ど作成したスレッド・プールを指定する必要があります。これにより、コネクタがインスタンス化されますが、そのコネクタはまだアプリケーションからは見えません。この手順を実行した後に、コネクタを設定に利用できるようになります。

手順3:コネクション・プールを作成します。リソース・アダプタの設定後、BucketStoreファクトリのためのコネクション・プールを作成する必要があります(図3参照)。

この手順では、以前に宣言した**jca-file-store**というコネクタから、コネクション・ファクトリを表す**BucketStore**ファクトリを選択します。**BucketStore**は**GenericManagedConnectionFactory**インスタンスでコネクション・ファクトリとして宣言されており、アプリケーション・サーバーによって適切に検出されます。参考までに、

LISTING 9a LISTING 9b

```
public class FileBucketTest {

    FileBucket cut;
    String directory = "./current";
    Closeable closeable;

    @Before
    public void initialize() {
        this.closeable = mock(Closeable.class);
        this.cut = new FileBucket(new PrintWriter(System.out),
            directory, this.closeable);
    }

    @Test
    public void autoClose() throws Exception {
        try (FileBucket bucket = new FileBucket(
            new PrintWriter(System.out), directory, this.closeable);) {
            bucket.begin();
        }
        verify(this.closeable).close();
    }

    @Test
    public void writeAndRollback() throws Exception {
        final String key = "hey";
        this.cut.begin();
        final byte[] content = "duke".getBytes();
        this.cut.write(key, content);
        byte[] actual = this.cut.fetch(key);
        assertThat(actual, is(content));
        this.cut.rollback();
        actual = this.cut.fetch(key);
        assertNull(actual);
    }
}
```

 [Download all listings in this issue as text](#)

GenericManagedConnectionFactoryで宣言されている@ConnectionDefinitionアノテーションをリスト13に示します。

コネクタのコネクション・プールを宣言した直後に、プールの設定を行う必要が

あります(図4参照)。

最適な値を予測することは困難であるため、まずはデフォルト値で実行し、ストレス・テストの実行後に設定を継続的にチューニングする方がはるかに簡単です。興味深いことに、GlassFish

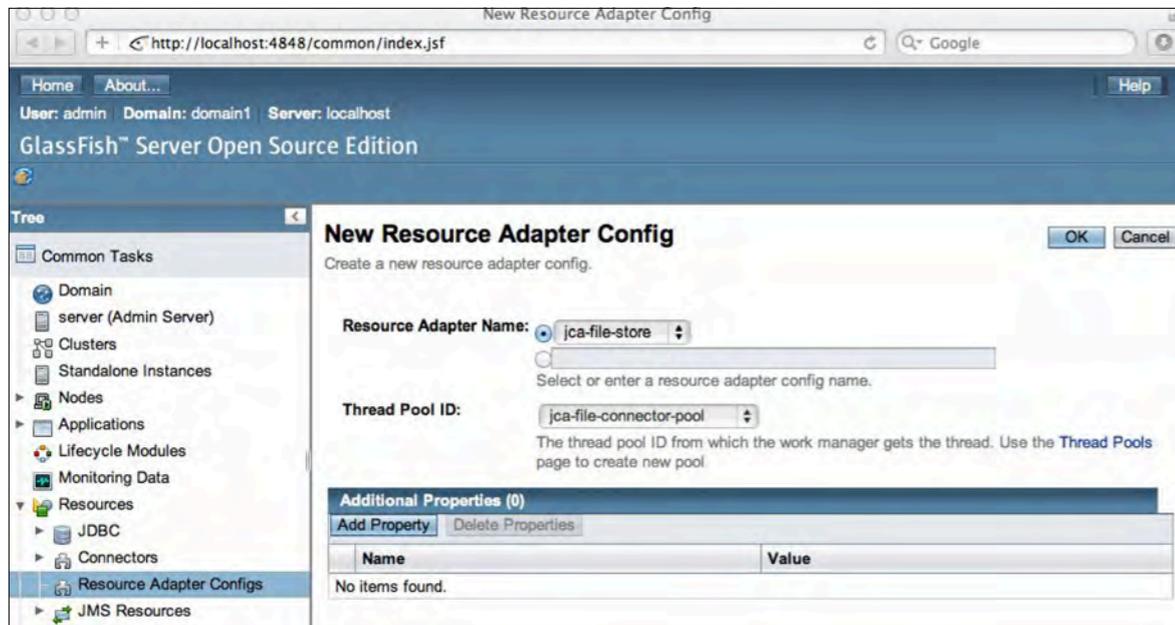


図2



図3

LISTING 10a LISTING 10b LISTING 10c LISTING 10d LISTING 10e LISTING 10f

```
import java.io.Closeable;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.util.List;
import javax.resource.ResourceException;
import static javax.resource.spi.ConnectionEvent.*;
import javax.resource.spi.*;
import javax.security.auth.Subject;
import javax.transaction.xa.XAResource;
public class GenericManagedConnection
    implements ManagedConnection, LocalTransaction,
    Closeable {
```

```
    private ManagedConnectionFactory mcf;
    private PrintWriter out;
    private FileBucket fileConnection;
    private ConnectionRequestInfo connectionRequestInfo;
    private List<ConnectionEventListener> listeners;
    private final String rootDirectory;
```

```
    GenericManagedConnection(PrintWriter out,String rootDir
    ectory,ManagedConnectionFactory mcf,
    ConnectionRequestInfo connectionRequestInfo) {
        this.out = out;
        this.rootDirectory = rootDirectory;
        this.mcf = mcf;
        this.connectionRequestInfo = connectionRequestInfo;
        this.listeners = new LinkedList<>();
        this.fileConnection =
    new FileBucket(out,this.rootDirectory,this);
    }
```

Download all listings in this issue as text

ではGenericManagedConnectionFactory内で宣言されているカスタム設定エントリも認識されます(リスト14参照)。

GenericManagedConnectionFactoryで公開されたrootDirectoryプロパティを使用して、Webベースの管理コンソールから直接、ルート・フォルダの場所を設定できます。setRootDirectoryメソッドのメタデータ(名前、パラメータの種類、@ConfigPropertyアノ

テーション、@Minアノテーションなど)から管理画面が生成されます。注目すべき点として、Bean Validationアノテーション(JSR 303)を使用して、ユーザーの入力値を検証し、無効なエントリを防ぐことができます。

手順4:JNDI名を指定します。最後の手順として、新しく設定した接続プールを独自のJNDI名で登録します(図5参照)。各種Java EEコンポーネントでは、ここで指定したJNDI名を使用

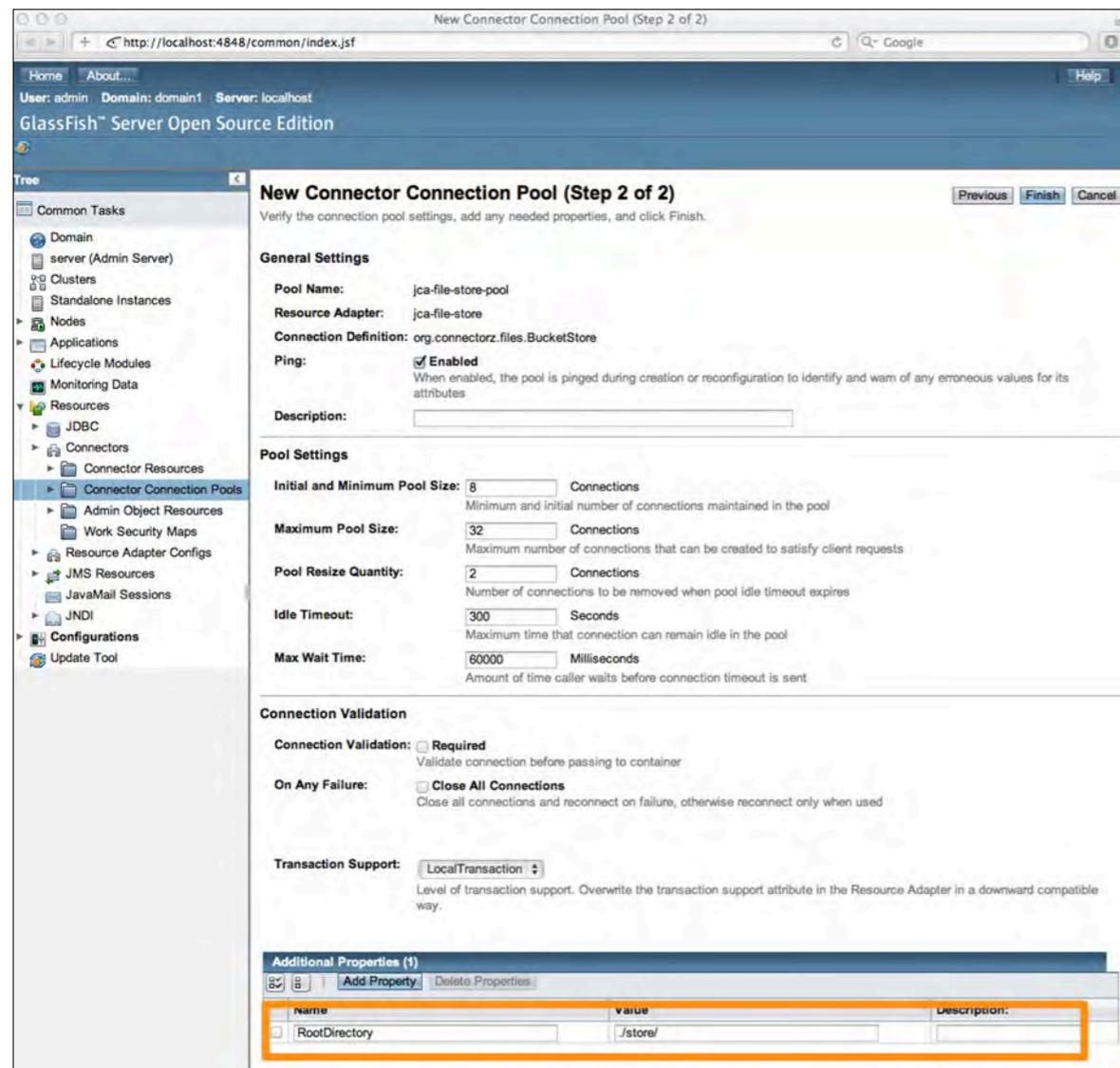


図4

LISTING 11a LISTING 11b LISTING 11c LISTING 12a LISTING 12b

```
import org.connectorz.files.BucketStore;
import java.io.PrintWriter;
import java.io.Serializable;
import java.util.Iterator;
import java.util.Objects;
import java.util.Set;
import javax.resource.ResourceException;
import javax.resource.spi.*;
import javax.security.auth.Subject;
import javax.validation.constraints.Min;
import org.connectorz.files.Bucket;
```

@ConnectionFactory(connectionFactory = BucketStore.class,

connectionFactoryImpl = FileBucketStore.class,
connection = Bucket.class,
connectionImpl = FileBucket.class)

```
public class GenericManagedConnectionFactory
    implements ManagedConnectionFactory, Serializable {
```

```
    private PrintWriter out;
    private String rootDirectory;
```

```
    public GenericManagedConnectionFactory() {
        out = new PrintWriter(System.out);
    }
```

@Min(1)
@ConfigProperty(defaultValue = "./store/",
supportsDynamicUpdates = true, description = "The
root folder
of the file store")

```
    public void setRootDirectory(String rootDirectory) {
        this.rootDirectory = rootDirectory;
    }
```

Download all listings in this issue as text

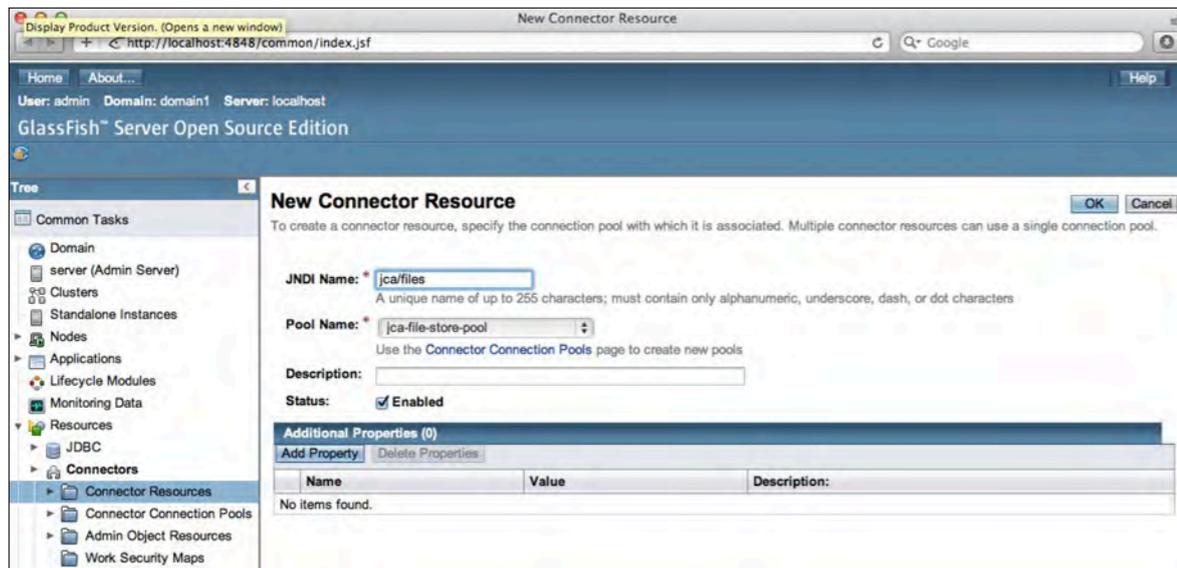


図5

して、BucketStoreインスタンスの依存性注入を行うことができます。

まとめ

カスタム値をすべて設定した後は、設定済みのJNDI名を使用して、ファイル・コネクタのリソースを各種Java EEコンポーネントにインジェクションできます。RESTfulリソースとして公開しているため、このJCAファイル・コネクタのすべての機能を任意のHTTPクライアントから利用できます。Javaクライアントの場合、FilesResourceを操作するもっとも簡単な方法は、標準化されたJAX-RS 2.0クライアントを使用することです(リスト15a、リスト15b参照)。

一般的に、JCAコネクタの実装では、クラス数を少なく抑え、XMLデプロイメント・ディスクリプタを使用せずに、Java EE環境内のファイルに「合法的に」アクセスできます。しかし、本来のメリットは、トランザクション型のセキュアで管理された監視可能な方法で、ファイルにアク

セスできることにあります。

さらに、リソース・アダプタのコネクション・プールを設定することで、公開されたリソースに対する操作の同時実行数を簡単に調整できます。リソース・アダプタの監視情報(コネクション・プール、ワーク・マネージャなど)も、アプリケーション・サーバーによって公開されます。GlassFishではRESTfulサービス経由で監視データを取得できるため、管理と監視が容易になります。</article>

LISTING 13 LISTING 14 LISTING 15a LISTING 15b

```
@ConnectionFactory(connectionFactory = BucketStore.class,
    connectionFactoryImpl = FileBucketStore.class,
    connection = Bucket.class,
    connectionImpl = FileBucket.class)
public class GenericManagedConnectionFactory{}
```

 [Download all listings in this issue as text](#)

LEARN MORE

- [connectorZ](#)
- [JSR 322: Java EE Connector Architecture 1.6](#)
- [JSR 303: Bean Validation](#)
- 『Real World Java EE Patterns—Rethinking Best Practices』の「Generic JCA」の章(press.adambien.com, 2011年)



VIKRAM GOYAL



Payment API—JSR 229入門

Java MEアプリケーション/ゲームでのJSR 229の利用法を学ぶ

アプリケーション内トランザクションを利用すれば、アプリケーションやゲームの利用者が、実行中のアプリケーション/ゲームを終了せずに決済処理をスムーズに実行できます。本記事では、Java MEアプリケーション/ゲームの開発者が利用できるPayment API(JSR 229)について取り上げます。Payment APIを実際にシミュレートしたサンプルアプリケーションを確認しながら、Payment APIの利用法を説明します。

注:本記事で紹介するサンプルアプリケーションのソース・コードは、NetBeansプロジェクトとしてこちらからダウンロードできます。

Payment APIの背景

JSR 229で定義されているPayment APIは、さまざまな解釈ができるように非常にシンプルに保たれています。このシンプルさがPayment APIの良い点です。Payment APIを利用することで、ゲーム/アプリケーション内

部からその下層の決済処理を呼び出すアダプタの作成が容易になります。

`javax.microedition.payment`パッケージ内に定義されたPayment APIには、2つのインタフェース、1つの具象クラス、4つの例外クラスが含まれます。

TransactionRecordインタフェースとTransactionListenerインタフェース

`TransactionRecord`インタフェースは、Payment APIの各種モジュールが実行するトランザクションの1レコードを表します。トランザクションのレコードの状態を表すために、整数のフラグを使用します。フラグの値は、`TRANSACTION_SUCCESSFUL`(正常終了)、`TRANSACTION_FAILED`(失敗)、`TRANSACTION_REJECTED`(拒否)のいずれかです。さらに、レコードに関するメタデータを提供する次の5つの情報取得メソッドが含まれています。

- `int getFeatureID()`—A フィーチャ(Feature)とは、料金を請求できる対象、あるいは支払いを必要とする対象を表します(ゲームのレベル・アップや付加価値サービスなど)。各フィーチャには一意の識別子を割り当てる必要があり、`getFeatureID()`メソッドは識別子の値(アプリケーションの開発者が決済モジュールに指定した値)を返します。各トランザクション・レコードは、1つのアプリケーション・フィーチャと関連付けられている必要があります。
- `int getTransactionID()`—トランザクション・レコードのプラットフォーム内で一意となる識別子を返します。
- `int getState()`—トランザクションの状態(正常終了、失敗、ユーザーによる拒否)を返します。
- `long getFinished`

`Timestamp()`—決済モジュールでこのトランザクションが認識された時刻を返します。

- `boolean wasMissed()`—現在のセッションではなく前のセッションで処理されたトランザクション・レコードであるかどうかを返します。

`TransactionListener`インタフェースは、トランザクションの処理完了(正常終了またはそれ以外)のタイミングを開発者に伝えるシンプルなりリスナー・インタフェースです。トランザクション・モジュールはトランザクションの処理を非同期的に実行し、処理の完了時にリスナーに通知します。`TransactionListener`インタフェースには`processed(TransactionRecord record)`というメソッドのみが定義されています。

シンプルのままにPayment APIは、さまざまな解釈ができるように非常にシンプルに保たれています。

TransactionModuleクラス

TransactionModuleクラスは、トランザクションの完了時(正常終了またはそれ以外)、TransactionListenerのprocessed(TransactionRecord record)メソッドを呼び出します。引数のレコードにトランザクションのすべてのデータと状態を保持します。

一言で言えば、TransactionModuleクラスは、アプリケーションとその下層の決済モジュールとを仲介するクラスです。process(int featureID, String featureTitle, String featureDescription)メソッドとprocess(int featureID, String featureTitle, String featureDescription, byte[] payload)メソッドが定義されており、アプリケーションでは、支払いの必要な新しいフィーチャについてユーザーに請求しようとするたびに、この2つのprocessメソッドを呼び出します。いずれのprocessメソッドも制御を即座に返します。その結果、前項で説明したように、決済モジュールによって登録されたすべてのリスナーに対してトランザクションの状態が送信されます。

され、ユーザーはフィーチャを購入することを確認できます。このGUIで購入を確認した後は、決済の正確性を検証するためのいくつかのフィールドの入力が続きます。

言うまでもなく、この画面が表示されるのは、いずれかのprocessメソッドに渡されたデータが有効である場合のみです。たとえば、featureTitleとfeatureDescriptionのいずれかがnullまたは空の場合、processメソッドはTransactionModuleExceptionをスローします。processメソッドを呼び出す前にsetListener()メソッドを呼び出してリスナーを登録していない場合は、processメソッドはTransactionListenerExceptionをスローします。また、後で説明するように、決済モジュールから参照されるJavaアーカイブ(JAR)マニフェスト・ファイルにこのフィーチャの情報がない場合は、processメソッドはTransactionFeatureExceptionをスローします。

process(int featureID, String featureTitle, String featureDescription, byte[] payload)メソッドは、byte[]配列形式のペイロードを引数として受け取り、この配列を使用してトランザクションの追加情報を決済処理アダプタに渡します。下層のアダプタでこのペイロードを処理できない場合は、TransactionPayloadExceptionがスローされます。

最終的に、これら2つのprocessメソッドはint型の一意識別子を返します。この識別子により、決済リクエストを特定して他のリクエストと区別できます。

TransactionModuleクラスにはコンストラクタが1つだけ定義されています。その唯一のコンストラクタであるTransactionModule(Object obj)では、通常はこのTransactionModuleクラスが実行されているMIDletの参照を引数に取ります。下層の決済モジュールの設定に問題がある場合は、このコンストラクタを呼び出すとTransactionModuleExceptionがスローされます。このコンストラクタが、すべての決済関連処理の起点となります。

TransactionModuleクラスにはdeliverMissedTransactions()メソッドも定義されています。deliverMissedTransactions()メソッドは、process()の呼び出しを開始してからリスナーのprocessed()メソッドを呼び出せる状態になるまでにアプリケーションがクラッシュした場合に使用されます。deliverMissedTransactions()メソッドはクラッシュによって消失したすべてのトランザクションを再生成することを試みます。

最後のメソッドはgetPastTransactions(int max)メソッドです。このメソッドは、現在のアプリケーションでこれまでに処理したTransactionRecordの配列を返します。

アダプタとフィーチャの準備

前項までに紹介したPayment APIに関する情報はいずれも重要ですが、アプリケーションでの決済処理の方法とタイミングに関する疑問をすべて解決するものではありません。Payment APIでは、実際の決済処理を単純に下層のアダプタに任せています。アダプタを提供する

のはデバイス・メーカーです。Premium-Priced SMSアダプタの場合もあれば、クレジットカードに請求するためのアダプタの場合もあります。

アプリケーション開発者がアダプタを制御することはほぼ不可能です。特定の決済アダプタが必要であると要求することはできますが、要求したアダプタを必ず利用できるわけではありません。要求したアダプタのいずれも利用できないデバイスや、あるいは要求したアダプタが正しく設定されていないデバイスでは、アプリケーションは動作しません。そのため、開発するアプリケーション/ゲームのターゲット・デバイスで決済アダプタがサポートされていることを、事前に十分に確認しておく必要があります。

一方で、フィーチャの準備については開発者が完全に制御できます。この準備は、MIDletアプリケーションのJARマニフェスト・ファイルで行います。本項では、下層のJava Application Descriptor(JAD)ファイルとJARマニフェスト・ファイルでのフィーチャの定義方法、およびフィーチャの決済への転換方法について説明します。

ここでは、「フリーミアム」モデルをベースとしたアプリケーションを作成することを想定します。フリーミアム・モデルとは、基本的なアプリケーションと一部のフィーチャを無料で提供し、特定の高度なフィーチャについては課金するという手法です。さらに、表1のような3つの価格プランを用意します。

PAYMENT APIについて
Payment APIでは、実際の決済処理を単純に下層のアダプタに任せています。

PLAN	PRICING
FREE	FREE
BUSINESS	\$5.00
PROFESSIONAL	\$10.00

表1

アダプタの定義

まず、JADファイルに少なくとも2つのエントリ(Pay-VersionとPay-Adapters)を定義する必要があります。

Pay-Versionは、JADファイルとJARマニフェスト・ファイルに関連付けるバージョンであり、通常は1.0です。

Pay-Adaptersは、アプリケーションと連携させるアダプタをカンマで区切って指定したリストです。指定したアダプタのうち少なくとも1つがターゲット・デバイスでサポートされている必要があります。まったくサポートされていない場合、アプリケーションは動作しません。アダプタの情報をJADファイルに定義する理由は、JADファイルがMIDletのインストール前にデバイスで読み取られるためです。これにより、デバイスでサポートされるアダプタがない場合には、デバイス側でそのMIDletのインストールを拒否できます。

最低限のエントリを含むJADのサンプルを次に示します。

Pay-Version: 1.0
Pay-Adapters: PPSMS, X-CCARD

PPSMSとX-CCARDは、Java MEツールキットでサポートされる決済アダプタの名称です。PPSMSは

Premium Priced SMSアダプタを指し、X-CCARDは非標準のクレジット・カード・アダプタのことです。Payment APIでは、Payment APIを実装するすべてのデバイスに、最低限PPSMSをサポートすることを義務づけています。

Payment APIの仕様では、デバッグやテストの目的でJADファイル内に含めることのできるその他のエントリも定義されています。

決済プロバイダの定義

前項で定義したとおり、1つのアダプタを複数のプロバイダと連携できます。たとえば、X-CCARD決済アダプタに対応する決済プロバイダとしては、Visa、MasterCard、American Expressが挙げられます。

各プロバイダを、JARマニフェスト・ファイル内でPay-Providers属性とPay-<ProviderTitle>-Info属性を使用して定義する必要があります。Pay-Providers属性に一覧で記述した決済プロバイダのそれぞれに対応するPay-<ProviderTitle>-Info属性が必要になります。これらのエントリを含むJARマニフェスト・ファイルの例を次に示します。

PLAN	PRICING	Pay-Feature-<n>/VALUE OF THIS ATTRIBUTE	Pay-<ProviderTitle>-Tag-<n>/VALUE OF THIS ATTRIBUTE
FREE	FREE	NOT REQUIRED	NOT REQUIRED
BUSINESS	\$5.00	Pay-Feature-0/0	Pay-ATT-Tag-0/5.00, 5550000, 500 Pay-VISA-Tag-0/5.00, 500
PROFESSIONAL	\$10.00	Pay-Feature-1/1	Pay-ATT-Tag-1/10.00, 5550000, 1000 Pay-VISA-Tag-1/10.00, 1000

表2

Pay-Providers: ATT, VISA
Pay-ATT-Info: PPSMS, USD, 999, 99
Pay-VISA-Info: X-CCARD, USD, VISA

このJARマニフェスト・ファイルは要するに、アプリケーションが決済プロバイダのAT&TおよびVisaと連携すること、AT&Tの決済ではPremium-Priced SMSアダプタ(JADファイル内で定義済み)を使用すること、Visaの決済ではクレジット・カード(同様にJADファイル内で定義済み)を使用することを示しています。

さらに、それぞれの決済プロバイダに対して、許容する通貨やその他の決済固有の情報を一覧で記述しています。AT&Tに対しては、Mobile Country Code(MCC)として999、Mobile Network Code(MNC)として99を指定しています。

決済プロバイダとフィーチャの関連付け
決済アダプタと決済プロバイダの定義が完了したので、次に、サンプルのフリーミアム・アプリケーションについて前述の価格モデルと、JARマニフェスト・ファイル内の関連するエントリとの相互関係

を設定します。この相互関係の設定のために、Pay-Feature-<n>属性とPay-<ProviderTitle>-Tag-<n>属性を使用します(言うまでもなく、無料の決済体系を定義する必要はありません。定義が必要となるのは、有料レベルの決済体系のみです)。

各フィーチャ(つまり、各決済)を各プロバイダ名にマッピングする必要があります。その際には、0から始まり1ずつ増加するタグ番号を含むタグを使用します。1つ目のフィーチャはPay-Feature-0であり、タグ番号0を使用して各プロバイダ名にマッピングする必要があります。詳細は表2を参照してください。

ビジネス・レベルにアップグレードするために\$5.00を支払うエンド・ユーザーに対しては、値0のPay-Feature-0属性を定義します。この値0を、先ほど定義した2つのプロバイダ名にマッピングします。デバイスでサポートされる決済アダプタやユーザーが選択する決済タイプを事前に把握することはできないため、これらの情報をプロバイダごとにPay-<ProviderTitle>-Tag-<n>属性を使用して次のように指定する必要があります。



Pay-Feature-0: 0

Pay-ATT-Tag-0: 5.00, 5550000, 500

Pay-VISA-Tag-0: 5.00, 500

A T & T タグでは、電話番号 5550000 に対して \$5.00 を請求する必要があること、500 が参照番号であることを示しています。Visa タグでは、\$5.00 を請求する必要があり、参照番号は 500 であることを示しています。

JAD ファイルと JAR マニフェスト・ファイルで指定する必要がある基本的な情報は以上です。後は、ユーザーに決済オプション (PPSMS または X-CCARD) を提示し、関連情報 (クレジット・カード情報の詳細など) の入力を求める処理を、デバイスに搭載された下層の決済アダプタに任せることとなります。

決済情報の更新

状況が変われば、決済オプションも変わります。インフレーションのために翌年にサービスの請求額を引き上げる場合もあれば、決済を受け入れる決済プロバイダの一覧を更新する場合もあります。Payment API では、次の 2 つの (必須) フィールドを指定することで、決済情報を簡単に更新できます。

Pay-Update-Stamp

Pay-Update-URL

Pay-Update-Stamp

フィールドは、JAR マニフェスト・ファイル内の決済情報が有効となる日付を表します。Pay-Update-URL フィールドは、更新された決済情報を定義する拡張子 .jpp のファイルに対するリンクです。この .jpp ファイルの形式は、JAR マニフェスト・ファイルと基本的に同じです。

この決済情報は、エンド・ユーザーが決済を行うたびに表示され、前回の決済日時と更新を利用できる可能性があることを通知します。エンド・ユーザーは更新が利用できるかどうかを確認してから決済処理に移り、更新された情報を使用して支払いを行うことができます。

サンプル・アプリケーション

サンプル・アプリケーションを実行する前に、理解しておくべき注意点について説明します。

オラクル製のエミュレータでは、バージョンごとに異なる決済アダプタが提供されています。オラクルの Sun Java Wireless Toolkit for CLDC 2.5.2 にはクレジット・カード・アダプタが付属していますが、同じくオラクルの Sun Java Wireless Toolkit for CLDC 3.0.5 にはクレジット・カード・アダプタはありません。そのため、クレジット・カード・アダプタの実際の動作を確認するためには、旧バージョンの Wireless Toolkit for CLDC 2.5.2 をインストールし、ターゲット・プ

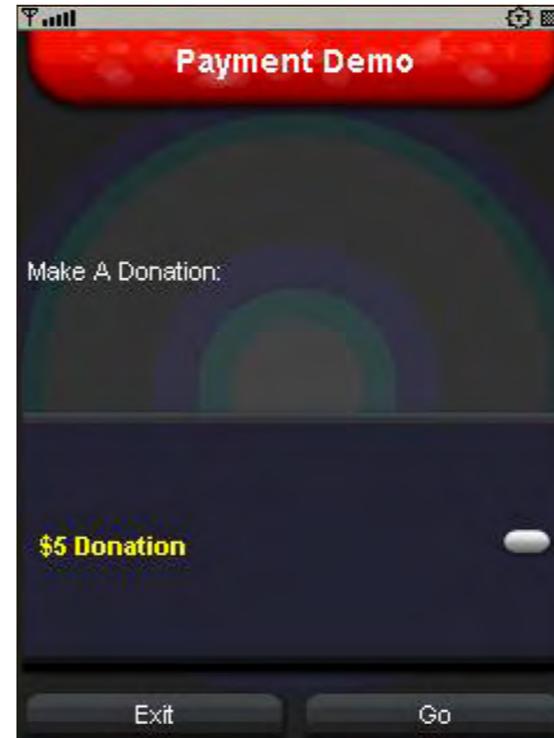


図 1

ラットフォームとして利用する必要があります。

また、PPSMS アダプタを使用する場合、正しい MNC コードと MCC コードを使用してターゲット・デバイスを設定する必要があります。サンプルの JAR マニフェスト・ファイルでは MNC コードに 999、MCC コードに 99 を使用しましたが、これらはターゲット・デバイスに存在するものと同じ値である必要があります。同じ値にする方法は、利用するツールキットによって異なります。バージョン 3.0.5 のツールキットの場合、メニュー・オプションの「Java ME」→「Device Selector」でこれらのコードの情報を確認し、更新できます。

本記事のサンプル・アプリケーションは非常にシンプルです。図 1 は、Lightweight User Interface Toolkit



図 2

(LWUIT) インタフェースを使用したスクリーンショットです。

このアプリケーションでは、ユーザーに「Vikram's Charity」への募金をお願いしています。2 種類の募金額、すなわち 2 つのフィーチャ (\$5.00 または \$10.00) が表示されています。ユーザーが (金額を選択してから) 「Go」を選択すると、アプリケーションでトランザクション・モジュールの process メソッドが呼び出され、Payment API が処理を引き継ぎます (リスト 1 参照)。

アプリケーションのメイン・スレッドは、決済処理が完了する (承認、拒否、失敗のいずれか) まで待機します。その後、トランザクション・モジュールの processed メソッドでコール・バックを受け取ります (リスト 2 参照)。

変更が容易

状況が変われば、決済オプションも変わります。翌年にサービスの請求額を引き上げる場合があります。Payment API では、2 つの (必須) フィールドを指定することで、決済情報を簡単に更新できます。





図3

processedメソッドでは最初に notify() を呼び出してメインスレッドを再開しています。最後に、トランザクションレコードの内容からトランザクションの状態(正常終了、拒否、失敗)を確認し、その状態に応じたメッセージを表示します。

図2は、決済を確認するための決済情報画面です。ここでは、更新データが表示されていることに注意してください。開発者がこの画面(や次の画面)の外観や操作性(ロック・アンド・フィール)を制御することはできません。画面の外観と操作性は下層のアダプタが制御します。このサンプルアプリケーションでは、バージョン2.5.2のツール

人気上昇中
Payment API
の普及は進んでおり、いずれは決済オプションを実装する際の一般的な選択肢となる可能性は高まっています。

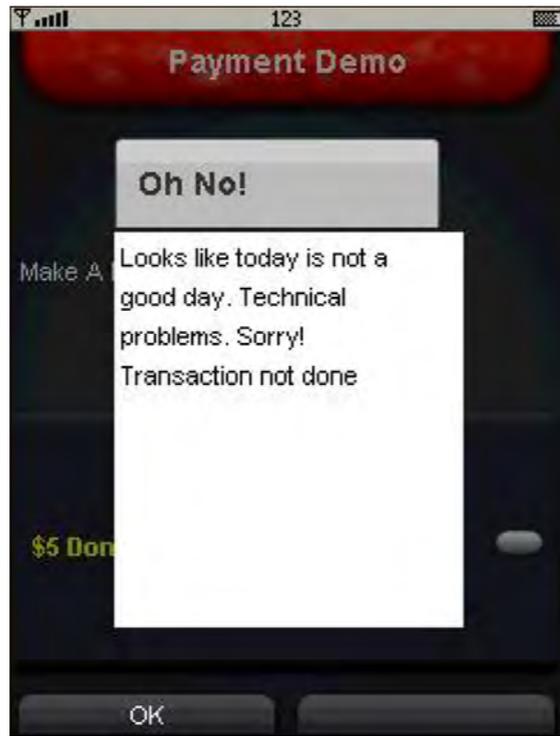


図4

キットを使用してクレジット・カードのトランザクションを表示しています。

図3は、図2でユーザーが「Yes」を選択した後に表示される、指定されたクレジット・カード情報の詳細入力画面です。

図3の画面でも、処理や画面の外観と操作性を開発者が制御することはできません。

図4は、技術的なエラーによって失敗したトランザクションの結果表示画面です。

最後に、図5と図6は、バージョン3.0.5のツールキットで、下層のアダプタとしてPPSMSを使用し、トランザクションが正常終了した場合のアプリケーションの画面を示しています。

LISTING 1 LISTING 2

```
try {
    int selectedId = selection.getSelectedIndex(); // what
    was selected
    // process it
    module.process(
        selectedId,
        "Donate " + (selectedId == 0 ? "$5.00" : "$10.00"),
        "Would you like to donate " + (selectedId == 0 ?
"$5.00" : "$10.00") +
        " to Vikram's Charity?");
    // and then wait... until we get notified that the
    // payment is processed
    synchronized(this) {
        try {
            wait();
        } catch(InterruptedException ie) {
        }
    }
} catch(Exception ex) {
    handleError(ex);
}
```

[Download all listings in this issue as text](#)

まとめ

現時点でPayment APIを実装しているデバイスはそれほど多くありませんが、Payment APIの普及は進んでおり、いずれは決済オプションを実装する際の一般的な選択肢となる可能性は高まっています。Payment APIは、決済の最終的な手順をデバイス・メーカーに任せるため、アプリケーションやゲームの利用者から

の資金調達のタスクが非常に容易になります。





図5

本記事では、JSR 229で定義された Payment APIについて紹介し、JAD ファイルおよびJARマニフェスト・ファイル内での決済アダプタ、決済プロバイダ、フィーチャの定義方法を説明しました。最後に、2種類の決済アダプタと2つのツールキットを使用した、Payment APIのシンプルな実例を示しました。
[</article>](#)

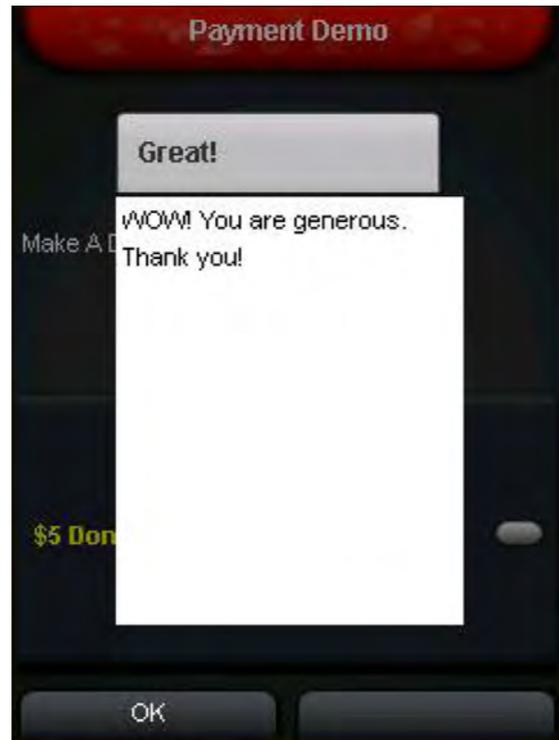


図6

LEARN MORE

- [JSR 229仕様](#)
- [ツールキットのダウンロード](#)

MAKE THE FUTURE JAVA

Java™

FIND YOUR JUG HERE

CEJUG is celebrating our 10-year anniversary in 2012! We follow Java technology with passion, share knowledge with pleasure, and create opportunities for students and professionals with ambition.

Hildeberto Mendonça
The Ceará Java User Group (CEJUG)

LEARN MORE

ORACLE®



TED NEWARD



パート2

Oracle Berkeley DB Java Edition の Java API

Oracle Berkeley DBの基本APIの動作を学習する

本シリーズの第1回では、Oracle Berkeley DBの基本を学習しました。

Oracle Berkeley DBは2つのレベルで構成されたデータストレージシステムです。非常に高いレベルでは、表面上何となくリレーショナルデータベースのように見えますが、実際には、「オブジェクトバイナリ」のような感覚でデータを管理できます。Oracle Berkeley DBの下層にあるのは「キー/値」形式の低レベルAPIです。基本APIと呼ばれるこのAPIでは、データの管理方法やデータストレージのカスタマイズ方法をより柔軟に指定できます。

前回の記事では基本APIについてほとんど触れることなく、低レベルのAPIであることと、他の高レベルのアプローチでは困難であるか不可能なくつかの興味深いソリューションを提供するという点のみを紹介しました。

基本API

基本的に、Oracle Berkeley DBの基本APIは、キー/値ストア（「キーバリュー・ストア」とも言います）をモデル化しています。つまり、それぞれが1つの巨大な表のようなデータベース（正確に言えばデータベース・オブジェクト）の中に、リレーショナルデータベースの用語で言えば「主キー」となる第1列と「バイナリ・ラージ・オブジェクト（BLOB）」となる第2列が存在します。

Oracle Berkeley DBの用語では、それぞれのキーと値のペアをレコードと言います。各レコードは `DatabaseEntry` クラスのオブジェクトであり、基本的に（内部では）バイト配列でしかありません。

これはまさに、基本APIが低レベルのアクセス方法であると説明したとおりです。

データベースを開く/閉じる

基本APIでデータベースを開く操作は、Direct Persistence Layer (DPL) から `EntityStore` を作成する操作とほとんど同じです。DPLは基本APIとは別のレイヤであり、第1回で既に詳細を説明しました。基本APIでもDPLの場合と同様に、`Environment` クラスと `EnvironmentConfig` クラスから操作を開始します。ただし、DPLのように `StoreConfig` を使用して `EntityStore` オブジェクトを作成するのではなく、今回は `Environment` クラスと `DatabaseConfig` オブジェクトのみを使用して `Database` オブジェクトを開きます（リスト1参照）。

今回も、`DatabaseConfig` オブジェクトはデータベースの動作を設定する手段として使用します。言うまでもなく、ソフトウェアにおける大原則（「開く必要のあるものは閉じなければならない」）がここでも成り立つため、開いたオブジェクトはすべて、次のように確実に閉じる必要があります。

```
@After public void close()
{
    db.close();
    dbEnv.close();
    dbDir.delete();
}
```

重要な点として、開いたオブジェクトを閉じていない場合、Oracle Berkeley DBから例外がスローされます。Java仮想マシンから `IllegalStateException: Unclosed Database:(データベース名)` という例外がスローされた場合、いずれかのオブジェクトを閉じ忘れていました。

一時データベース

UNIXの世界ではOracle Berkeley DBデータベースは幾度となく配備されましたが、その用途は「永続化」ではなく、プログラム実行中の一時的なデータを保持する一時データストアでした。言い換えれば、プログラムの終了後、この一時データストア内のデータは無関係で意味のないものとなります。むしろ、次回の実

**簡単
他のデータベース・システムと比較して、基本APIを使用した読み取りや書き込みは本当に簡単です。**

行時にデータが残存している場合は、そのデータが原因で問題が発生することもありました。(入力を読み取り、中間結果を計算し、その中間結果をディスクに保存してメモリ内に中間結果のすべてを保持しないようにして、さらに結果を計算し、最終結果を別の場所に書き込むプログラムがあるとしたら、その場合、ディスクに書き込まれる中間結果は、プログラムの次の実行時に混乱をきたす原因となります。)

こうした不要なデータがプログラムの実行後に残存することを防ぐために、Oracle Berkeley DB APIにはデータベースを一時的なストアとして設定するオプションがあります。このオプションを使用した場合、プログラムの終了時に、データベースを表す全ファイルが完全に削除されます。(正確に言えば、システムでディスクへの書き込みを回避できるのであれば、その回避を徹底することによって、データベースは本質的にインメモリデータベースとなります。しかし、ディスクI/Oがまったく発生しないという想定は安全ではないということがOracle Berkeley DB APIのドキュメントに明記されています。すべてのデータをキャッシュ内に維持できない場合、一部のデータがディスクに書き込まれることになり、書き込まれたファイルは後で削除されます。)

データをディスクに永続化することについてシステムで考慮する必要がないため、一時データベースは驚くほど高速に動作します。ただし、このメリットと引き換えに、クラッシュの発生時にはデータが完全に消失するという明らかなデメリットも生じます。データベースを一時的データベースとして設定するた

めには、データベースを開くために使用するDatabaseConfigオブジェクトのsetTemporary(true)を1回呼び出す必要があります。ただし、注意すべき点があります。それは、コード上では実際にディスクに書き込まない場合でも、データベース構成下にあるディスクは重要であるということです。たとえば、以前に一時的ではないデータベースを保存していたディレクトリ上でEnvironmentを開く場合、Oracle Berkeley DBでは前述とは別のIllegalStateExceptionがスローされます。この例外は、過去に一時データベースではなかったデータベースを一時的データベースに設定することはできないという内容を示します。

データベースの読取りと書込み

他のデータベース・システムと比較して、基本APIを使用した読取りや書込みは非常に簡単です。前述のとおり、基本APIはバイト配列型のキー/値ストアであるため、データを保存するためには、リスト2のようにキーのバイト配列と値のバイト配列を指定してputを呼び出します。

繰り返しになりますが、これはまさに、基本APIが低レベルのアクセス方法であると説明したとおりです。

putの1つ目のパラメータはTransactionオブジェクトです。このパラメータについては、現時点では(トランザクション非対応型にして)説明を簡略化するため無視できます。さらに、基本APIの重要なポイントを強調する目的で、Databaseオブジェクトのcount()メソッドも呼び出しています。count()メソッドは、データベース全体に含まれるすべてのキーと値のペアの個数を返し

LISTING 1 LISTING 2 LISTING 3 LISTING 4

```
File dbDir = new File("./data");
Environment dbEnv;
Database db;
@Before public void open()
{
    if (!dbDir.exists())
        dbDir.mkdir();

    EnvironmentConfig config = new EnvironmentConfig();
    config.setAllowCreate(true);
    dbEnv = new Environment(dbDir, config);

    DatabaseConfig dbConfig = new DatabaseConfig();
    dbConfig.setAllowCreate(true);
    db = dbEnv.openDatabase(null, "database", dbConfig);
}
```

 [Download all listings in this issue as text](#)

ます。前述のとおり、このデータベースは基本的には、キーと値のペアによる1つの巨大なコレクションです。したがって、リスト2では1つのレコードを書き込み、次にデータベースがちょうど1つのレコードを保持していることをassert文で確認しています。

データベースからキーと値のペアを読み取る方法は容易に予想できます(リスト3参照)。

get呼び出しの1つ目のパラメータもTransactionオブジェクトであり、最後のパラメータはLockModeの列挙値です。LockModeの列挙値は、同時実行性に関して使用するロックの種類を表します。nullを指定すると、データベースを開いたときのDatabaseConfigによるデフォルト設定が使用されます。

リスト4のように、データベースからキーと値のペアを削除する操作も、同じく

容易に予想できます。さらに、予想どおり、`delete`呼び出しの1つ目のパラメータもやはり`Transaction`オブジェクトです。

ところで、トランザクションを使用して処理を実行する場合は、`Environment`から`Transaction`オブジェクトを1つ取得し、これまで紹介したそれぞれの呼び出しの1つ目のパラメータに渡します。トランザクションについては本記事の対象範囲から少し外れますが、リレーショナルデータベースのトランザクションに精通している開発者にとって、`Transaction`オブジェクトのAPIは非常に理解しやすくなっています。

データベースの遅延書込み

基本APIの(そして実際のところはOracle Berkeley DB全体の)おもな利点の1つが処理速度です。従来型のSQLデータベースの操作とOracle Berkeley DBデータベースの操作を比較した場合、ほとんどの操作でOracle Berkeley DB APIの方が高速です。ただし、多くの場合、この速度を達成するために何らかの妥協をしています(たとえば、インメモリデータベースに関する前述の説明を参照してください)。

Oracle Berkeley DBデータベースでパフォーマンスをさらに向上する1つの方法として、ディスクへの書込みを`put`の呼び出し時に行うのではなく、適当なタ

イミングまで遅らせるというものがあります。この処理は遅延書込みと呼ばれています。データの永続性に関する要件がそれほど厳しくないシステムの場合、遅延書込みによってパフォーマンスが大幅に向上します。

遅延書込みモードでデータベースを開くためには、`DatabaseConfig`オブジェクトの`setDeferredWrite(true)`を呼び出します。この呼び出しは、`DatabaseConfig`オブジェクトを使用して`Database`オブジェクトを作成する前に実行します。この設定によって、データベースに対するすべての書込みが、開発者が`Database`オブジェクトの`sync()`を呼び出して強制的にデータを書き込むまで引き延ばされます(このデータ書込み

の強制はチェックポイントと呼ばれ、同時実行性の文脈ではバリアと呼ばれることもあります)。

パフォーマンスが向上する理由は簡単です。開発者が`sync()`を呼び出すまでディスクへのすべての書込みを遅らせることで、Oracle Berkeley DBでは書き込まれたばかりのレコードに対する更新をメモリ内で実行できます。ディスク内をシークして書き込む必要はなく、更新を反映した最終結果のみを後でディスクに書き込むことができます。

たとえば、次のような一般的なエンド・ユーザーのシナリオで遅延書込みを行う利点を検討します。ユーザーが

レコードを1つ作成し、そのレコードを編集し、さらに同じレコードを編集し、その後そのレコードが不要だと気づいて削除するというシナリオです。この場合、遅延書込みでは、開発者が最後に`sync()`を呼び出したときには何も書き込む必要はなく、呼び出しを即座に完了できます。ただし言うまでもなく、このメリットの裏には、`sync()`を呼び出してから次に呼び出すまでの間にデータが消失するというデメリットもあります。

なお、お気づきかも知れませんが、一時データベースは常に遅延モードで書き込まれます。

タプルとバインド

データをバイト配列として保存する手法は強力ですが、操作が非常に難しくもあります。確かに、Javaのオブジェクト・シリアライズ機能ですべてのオブジェクトをバイト配列に変換し、またバイト配列を元に戻すことはできます。しかし、シリアライズには、他の方法と比較して非常に時間がかかるという欠点があります。(理由の1つは、型情報がシリアライズ後のバイト配列内にエンコードされることです。そのため、型情報を常に読み書きする必要があるだけでなく、必要なストレージ容量も増加します。)したがって、Oracle Berkeley DBでは、データ要素のコレクションを保存(および変換)するための別の方法が用意されています。それは、`TupleBinding`とBIND APIです。

タプルとは、概念的には軽量のオブジェクトのようなものであり、名前が関連付けられていないデータ要素のコレクションです(名前が関連付けられているデータ要素は、オブジェクトのフィール

ドとなります)。タプルは、非常に軽量なデータ転送オブジェクトまたはパラメータオブジェクトと考えることができます。

Oracle Berkeley DBでは、`TupleBinding`を使用して、多様な要素をどのように1つのレコード内に配置するのかを表すことができます。そのためには、カスタムの`TupleBinding`実装を作成し、読取りと書込みを実行します(**リスト 5a, 5b**参照)。`TupleBinding`は、用途が限定され、開発者側での実装作業がより必要な`Serializable`の一種と言えます。

`TupleBinding`のコード例は少し長いのですが、一見した印象ほどには難しくありません。このコード例では、`RSSPost`オブジェクトを保存しようとしています。`RSSPost`オブジェクトは基本的には第1回の`BlogPost`オブジェクトと同じであり、`String`型のタイトル、`String`型のテキスト、`Date`型の投稿日が含まれます。この`RSSPost`オブジェクトを、ブログ記事のタイトルをキーとして保存します。そのためには、キーと値のそれぞれに対する`TupleBinding`オブジェクトが必要です。

キーを表すオブジェクトは、`TupleBinding`に「プリミティブ」な`String`型の`TupleBinding`インスタンスを問い合わせ、簡単に取得できます。値を表すオブジェクトを取得するコードは、これよりも手がかかります。Oracle Berkeley DBには`RSSPost`オブジェクト用の組込みのバインディングが存在しないため、開発者側で`TupleBinding`インタフェースを実装してバインディングを作成する必要があります。(通常は、`RSSPost`オブジェクト自体に対してこのような実装を行うか、独立したクラス

非常に高速

データをディスクに永続化することについてシステムで考慮する必要がないため、一時データベースは驚くほど高速に動作します。ただし、このメリットと引き換えに、クラッシュの発生時にはデータが完全に消失するという明らかなデメリットも生じます。

を作成します。しかし、このコード例では、自己完結型のコードにするために、匿名の内部クラスとして実装しました。）

このインタフェースの実装は、次の2つのメソッドで構成されます。

- 対象のオブジェクトを引数で受け取り、そのオブジェクトを `TupleOutput` オブジェクト(ディスクに書き込むためのオブジェクト)に書き込むメソッド
- `TupleInput`(ディスク上のデータを表すオブジェクト)を引数で受け取り、すべてのフィールドをまったく同じ順序で抽出した `RSSPost` オブジェクトを返すメソッド

これらの2つのバインディング・オブジェクトの準備が完了したら、後は前述のとおり `get()` と `put()` を実行するだけです。

検索

これまでに説明したキー/値形式はとても有能ですが、それは開発者が検索対象のキーを知っている場合に限られます。キーを知らない場合や、ある範囲のデータを取得したい場合は、キー/値形式はあまり便利ではありません。しかし幸いなことに、Oracle Berkeley DBでは `Cursor` APIがサポートされており、調査対象の値をレコード全体から検索できます。

理論的には、`Cursor` APIは `for` ループに非常によく似ています。データベースから `Cursor` を取得し、取得した `Cursor` でキーに基づいた検索を実行し、その後調査対象のキーと値のペアを検索するためのループを実行します(何も指定せずに反復処理を開始した場合には、処理はデータベースの最初のレコードから始まり、すべてのレコードが返されます)。しかし実際には、`Cursor` の動作は思った

よりも少し手が込んでいます。

まず、データ・セットを少し操作しやすくするために、重複レコードをサポートするようデータベースを構成します。つまり、キーが同じで値の異なる複数のレコードをデータベースに正常に保存できるようにします。そのためには、`DatabaseConfig` オブジェクトを渡して `Database` インスタンスを取得する前に、`DatabaseConfig` オブジェクトの `setSortedDuplicates(true)` を呼び出す必要があります。

この設定を完了している場合に、**リスト6**のように、非常に容易に複数のレコードをデータベースに挿入できます。

次に、`Cursor` を取得するために、データベースに対して `Cursor` を1つ取得するように問い合わせます。ただしこの際に、`Cursor` に対して、あるキーを基準として検索を実行するように指定する必要があります(ここでも `DatabaseEntry` オブジェクトの形式を使用します)。**リスト7**のように、このキーを、値を保持する `DatabaseEntry` とともに渡す必要があります。

`Cursor` では、検索条件に合致するレコードの数が認識され、このレコード数の情報は `count()` メソッドで取得できます。そのため、「Ted」をキーとする6レコードがすべて見つかったかどうかを容易にテストできます。

ただし、結果の反復処理は少し手が込んでいます。 `getNext()` または `getPrev()` を呼び出したときに渡される `DatabaseEntry` オブジェクトには、対応する値が格納されています。しかし、最初のキーと値のペアは既に `cursorKey` と `cursorValue` にフェッチされていま

LISTING 5a LISTING 5b LISTING 6 LISTING 7 LISTING 8 LISTING 9

```
@Test public void storeReadAndRemoveATuple()
    throws java.io.UnsupportedEncodingException
{
    RSSPost post = new RSSPost("The Vietnam of Computer
        Science",
            "Blah blah blah...");

    TupleBinding<String> stringBinding =
        TupleBinding.getPrimitiveBinding(String.class);

    TupleBinding<RSSPost> rssbinding =
new TupleBinding<RSSPost>() {
    public void objectToEntry(RSSPost post, TupleOutput
to) {
        to.writeString(post.getTitle());
        to.writeString(post.getText());
        to.writeString(post.getPostingDate().toGMTString());
    }
public RSSPost entryToObject(TupleInput ti) {
    RSSPost post = new RSSPost();
    post.setTitle(ti.readString());
    post.setText(ti.readString());
    post.setPostingDate(new Date(ti.readString()));
    return post;
}
};
```

 [Download all listings in this issue as text](#)

す。そのため、少し奇妙なループ処理に見えます(**リスト8**参照)。

他のほとんどのOracle Berkeley DBオブジェクトと同様に、`Cursor` インスタンスも確実に閉じる必要があります。閉じ忘れた場合は、`IllegalStateException`が発生します。

不思議なことに、JUnitテスト結果のテキスト・ファイル(**リスト9**参照)では、すべての値が辞書の順番に並んでいます。Oracle Berkeley DBではデフォルトで、単純なバイト配列の値の比較によってレコードがソートされます。本記事のコード例では、このソートが上手く

作用しています。ソート順を変える場合は、カスタムの [Comparator](#) を作成し、使用するデータベースに渡すことができます。

まとめ

明らかに、基本APIの操作はDirect Persistence Layerの操作ほど簡単ではありませんが、DPLでは実現できなかった低レベルの柔軟性を確実に得られます。基本APIとDPLの両方があるため、同じデータベースで2つのニーズに対応できるという利点があります。1つはキー/値ストア(CassandraやBigTableに部分的に類似)として使用できること、もう1つはオブジェクト・ストレージ・データベース(db4oやVersantなどのオブジェクト指向データベース管理システムに類似)として使用できることです。

基本APIが高レベルAPIよりも望ましい点として、元々のCベースのOracle Berkeley DB形式との互換性が挙げられます。この互換性により、C/C++によるUNIXレガシー・アプリケーション(正直なところ、ほとんどのMicrosoft WindowsアプリケーションではOracle Berkeley DBの存在が知られておらず、Oracle Berkeley DBを利用して出荷されたMicrosoft Windowsアプリケーションはかなり少ないのです)をJavaに移植する必要がある場合、基本APIを使用すれば、コード内である種の明示的な変換処理を実行しなくても移植を行うことができます。

大幅な性能向上
データの永続性に関する要件がそれほど厳しくないシステムの場合、**遅延書込みによってパフォーマンスが大幅に向上**します。

本シリーズではOracle Berkeley DBについて説明してきましたが、まだ知るべきことは多くあります。しかし、とりあえずはこのあたりで締めくくります。幸いにも、オラクルでのOracle Berkeley DB開発者は、Oracle Berkeley DBの文書化にも力を入れています。そのため、すぐにでも詳細を学びたい場合は、[Oracle Berkeley Databaseのドキュメントの](#)

ページを参照してください。

とにもかくにも、Oracle Berkeley DBというデータベースが存在すること、今すぐ入手して使用できることをお伝えすることが本記事の最終目標です。本記事でOracle Berkeley DBを知った皆さんに、1980年代の古いアニメ番組で毎週のように伝えられた言葉を贈ります。「知った者が勝負を制する(Knowing is half the battle)」のです。 [</article>](#)

LEARN MORE

- [Oracle Berkeley DB](#)

MAKE THE FUTURE
JAVA

Java™

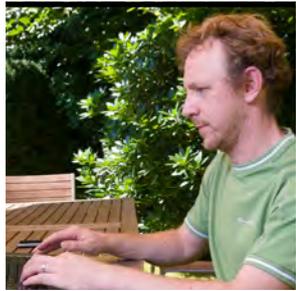
FIND YOUR JUG HERE

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.

Regina ten Bruggencate
JDuchess

LEARN MORE

ORACLE®



JOHAN VOS



DataFX: 実世界データを JavaFXコントロールに展開

データを取得、解析し、さまざまなJavaFXコントロールに表示するDataFX

JavaFX 2プラットフォームでは、さまざまなUIコントロール(ListView、TableView、TreeView、Chartなど)で、データをわかりやすく魅力的に表現できます。JavaFX 2 APIには、データの表示、変更内容の自動的な視覚化、およびユーザー入力の取得のための手法が数多く用意されています。これらの手法は従来型のCRUDアプローチであり、リモートにあるデータソースのレコードの作成、読取り、更新、削除を行うためにほとんどのアプリケーションで必要とされます。しかし、大多数のアプリケーションでは、このアプローチを採用した場合、非常に多くの定型的なコードを記述しなければなりません。

ドメイン固有のビジネス・アプリケーションの開発者は通常、関連データを取得するための煩雑なコーディングに時間を割くのではなく、ドメイン固有のロジックに集中して、ドメイン固有の機能をもっとも効果的な方法で利用者に提供したいと考えています。また、

データソースで必要とされる定型コードは書き間違いやすい傾向にあります(そのため、後で隠れたバグの攻撃に苦しむこととなります)。データの取得、解析、処理、表示を行うコードは、高性能で、スレッドセーフで、柔軟である必要があります。

幸いにも、これこそ、BSDライセンスで配布されているオープンソースのDataFXプロジェクトが取り組んでいる分野です。DataFXにより、さまざまなソースから異なる形式のデータを取得して解析し、各JavaFXコントロールに表示する処理が容易になります。

DataFXは大きく分けて2つの相補的な要素で構成されています。

- **データソース:** データの取得と解析のツール
 - **セル・ファクトリ:** 多くのJavaFXコントロールにデータを容易に表示するツール
- この2つの構成要素を組み合

わせることで、データの取得と解析を行い、使いやすい上に非常に強力なAPIに表示するというエンド・ツー・エンドの処理が完成します。

それでは、この2つの領域について深く掘り下げていきます。

注:本記事で紹介するサンプルのソースコード一覧は[こちら](#)からダウンロードできます。

データソース

DataFXの第1の構成要素は、データの取得と解析に焦点を置きます。また、第2の構成要素はセル・ファクトリに焦点を置き、解析後のデータを各コントロール上で視覚化する処理に取り組んでいます。セル・ファクトリについては本記事の後半で取り上げます。

Javaプラットフォームには既

**より簡単に
DataFXにより、
さまざまな
ソースから異なる
形式のデータを
取得して解析し、
各JavaFX
コントロールに
表示する処理が
容易になります。**

に、複数の異なるソースからデータを取得できる数多くのAPIと、そのための便利なアプローチを実現する豊富なライブラリが備わっています。

JavaFXプラットフォームでリモート・データを取り扱う際の重要事項として、次の2点に着目してください。

- **Observable** インタフェースと**ObservableList** インタフェース:この2つのJavaFXインタフェースを使用することで、データの変更をすぐに関心のあるリスナー (UIコントロールを含む)に通知できます。データの取得には、ブロックする一度の**get**呼び出しではなく、**ObservableList**を使用して、利用可能になったデータを追加していくようにする必要があります。また、この間、ユーザー・インタフェースは応答可能な状態を維持する必要があります。実際、多くのUIコントロールでは、基盤となるデータソースを監視するリスナーが自動的に追加され、データの変更時には表示が自動的に更新されます。
- スレッド:シーン・グラフを変更する操作はすべて、JavaFXアプリケーション・スレッドから開始する必要があります。そのため、**ObservableList**へのデータ・エントリの追加は、JavaFXアプリケーション・スレッドから実行する必要があります。一方で、データの取得中に、ユーザー・インタフェースが応答不可能な状態にならないようにする必要があります。JavaFXの**Worker**インタフェース、**Service**クラス、**Task**クラスでは、

このスレッド機能に関する興味深いオプションが用意されています。

JavaFXコントロールに表示されるデータの出所、プロトコル、および特性はさまざまです。DataFX APIでは、これらの特徴を明確に区別するために、次の2つの概念を提供しています。

- **org.javafxdata.data-sources.reader** パッケージ (**DataSourceReader**インタフェースと複数の実装クラス、ビルダーを含む)
- **org.javafxdata.datasources.provider** パッケージ (複数の**DataSource**プロバイダ・インタフェース、実装クラス、ビルダーを含む)

DataSourceReaderはデータの出所(ファイル、ネットワーク・リソース、単純なJavaオブジェクトなど)を抽象化したものです。一方、**DataSource**はデータのプロトコル(Javascript Object Notation[JSON]、XML、Javaオブジェクトなど)とデータの宛先(**ObservableList**に設定されるPlain Old Java Objects[POJO]や、**TableColumn**インスタンスに設定されるString値など)を抽象化したものです。

DataSourceReader、プロバイダ、コントロールの関係を図1に示します。

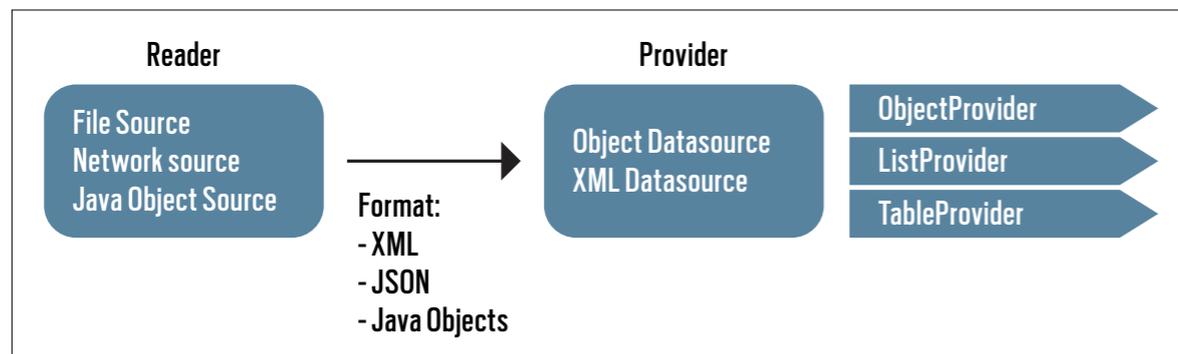


Figure 1

LISTING 1

```
RestRequestBuilder rrb = new RestRequestBuilder
("http://search.twitter.com").path("search.rss").queryParam
("q", "javafx");
DataSourceReader dataSourceReader = rrb.build();
```

Download all listings in this issue as text

これ以降、小さなアプリケーションを作成しながら図1の流れを説明していきます。このアプリケーションでは、Twitter APIを使用して「JavaFX」という語を含むツイートを問い合わせ、見つかったツイートを**ListView**と**TableView**に表示します。

DataSourceReader: **DataSourceReader**は、**DataSource**が使用する**InputStream**を提供するインタフェースです。**DataSourceReader**のそれぞれの実装クラスで、データを取得し**InputStream**を作成する独自の方法を実装します。**DataSource**は**InputStream**の出所を考慮する必要はありません。

現在、DataFXには次の3種類の**DataSourceReader**実装クラスが含まれています。

- **FileSource**: ファイル・システムからのデータの読取り
- **NetworkSource**: ネットワークからのデータの読取り
- **JavaObjectSource**: Javaオブジェクトを含む**ObjectInputStream**を提供
開発者はこれらの実装クラスを直接使用できます。

DataFXには**RestRequestBuilder**というクラスがあります。開発者はこのクラスのBuilderパターンを使用して、Java API for RESTful Web Services (JAX-RS)の開発者にはお馴染みの方法で**NetworkSource**を作成できます。実際、現在のビジネス・アプリケーションの多くが、Representational State Transfer (REST) サービスからJSON形式またはXML形式でデータを取得しています。**RestRequestBuilder**を利用すれば、RESTエンドポイントに対して呼び出しを行い、その結果のデータを取得できます。

リスト1は、Twitter検索APIを呼び出してデータを取得し、そのデータを提供する**NetworkSource**を作成するコードです。

RestRequestBuilderのコンストラクタは、RESTエンドポイントのホスト名をパラメータとして受け取ります。次に、**path()**メソッドを呼び出してエンドポイントの完全なパスを追加しています。**path()**メソッドは複数回呼び出すことも可能です。その場合、呼び出すたびにパスの一部が追加されます。その後、**queryParam()**メソッドを使用して問合せのパラメータを指定できます。また、**formParam()**メソッドを使用してフォー

ムのパラメータを指定できます。

デフォルトでは、`RestRequest Builder`によって作成される `NetworkSource`でGET操作を使用しますが、`method()`メソッドでPOST、PUT、DELETEのいずれかの操作を指定することもできます。

次に、`RestRequestBuilder`の `build()`メソッドを呼び出して、`NetworkSource`を作成します。この段階では、リモート・エンドポイントにはまだ接続しません。`NetworkSource.getInputStream()`メソッドを呼び出すことで、RESTエンドポイントに対する接続が開始されます。

`org.javafxdata.datasources.provider`パッケージ内の各種実装クラスでは、`DataSourceReader.getInputStream()`メソッドで取得した `InputStream`を使用しています。これらのクラスは、`InputStream`の出所はまったく意識しません。そのため、XMLやJSONのデータを含む `FileSource`をテスト中に利用し、REST呼び出しによって取得した同様のXMLやJSONのデータを含む `NetworkSource`を本番環境で利用することも簡単です。

ツイートを含む「テスト・ファイル」は、Curlなどのコマンドライン・ツールで簡単に作成できます。Linuxでは、JavaFX

に関するツイートを含むファイルを **リスト2**のように作成します。

これで、`mytweets.xml`ファイル `FileSource`の入力として利用できるようになります (**リスト3**参照)。

`DataSourceReader`の各種実装クラスは提供するデータの形式には依存しませんが、データ形式を判別できる必要はあります。そのため、これら実装クラスのコンストラクタでは、期待される応答データの形式を指定できます。また、`RestRequestBuilder`は、応答データの形式 (XML、JSON、またはJavaオブジェクト)を指定するための `format()`メソッドを持っています。

DataSourceプロバイダ: JavaFXコントロールにデータを設定するための次の手順は、データを解析し、`ListView`と `TableView`の各項目にツイートを追加することです。これらの手順では、`DataSourceReader`で取得したデータを使用します。先ほど説明した2つの

アプローチ (RESTエンドポイントから取得したデータを含む `NetworkSource`と、ファイル・システム上の静的なデータを含む `FileSource`)でこれらの手順に違いはありません。

`org.javafxdata.datasources.provider`パッケージには、多数のインタフェース、実装クラス、およびビルダーが

コード・ヘルパー
データの取得、解析、処理、表示を行うコードは、**高性能で、スレッドセーフで、柔軟である必要**があります。幸いにも、これこそが BSD ライセンスで配布されているオープンソースの DataFX プロジェクトが取り組んでいる分野です。

LISTING 2 LISTING 3

```
curl "http://search.twitter.com/search.rss?q=javafx" > /tmp/mytweets.xml
```

Download all listings in this issue as text

含まれています。ある特定のコンテキストでもっとも効果的な実装クラスを判断する基準は次のとおりです。

- 入力プロトコル: `ObjectDataSource` クラスは、XML、JSON、シリアライズされたJavaオブジェクトに対応しています。`ObjectDataSource`クラス以外の実装クラスでは、その他のプロトコルに対応している場合もあります。
- 出力プロトコル: 実際のJavaオブジェクトを操作したいと考える開発者もいれば、XMLの処理を好む開発者もいます。`ObjectDataSource`は、入力データを実際のJavaオブジェクトに変換します。一方、`XmlDataSource`は入力データのXML文書を維持し、Javaオブジェクトに変換せずにXMLを直接解析することで、JavaFXコントロールに必要な値を提供します。
- コンシューマ:
 - `ObjectProvider`を実装する `DataSource`は、監視可能な汎用データを提供します。
 - `ListProvider`を実装する `DataSource`はこの `ObjectProvider`を継承しており、オブジェクトの `ObservableList`を提供します。そのため、`ListView`にデータを設定する場合に適切なインタフェースです。
 - 最後に、`TableProvider`を実装す

る `DataSource`は `ListProvider`を継承しており、列情報の設定メソッドや取得メソッドが追加されています。そのため、`TableView`のデータ設定に必要なクラスです。

本記事のTwitterのサンプルを作成するためには、(Twitterでサポートされる)XMLまたはJSONを処理できる実装クラスが必要です。このサンプルではXML形式を利用しますが、JSONも同じく簡単に利用できます。

データのコンシューマは `ListView`と `TableView`のいずれかです。ここからは、`ObjectDataSource`を使用した `ListView`または `TableView`の実装がいかに容易であるかを説明します。`ObjectDataSource`は `TableProvider`を (それにより `ListProvider`も)実装しており、XMLおよびJSONと連携できます。

`org.javafxdata.datasources.provider`パッケージ内の `ObjectDataSourceBuilder`は、`ObjectDataSource`の作成に役立つ便利なクラスです。

リスト4のコードでは、`ObjectDataSource`を作成し、入力されたツイートを `ObservableList allTweets`に設定しています。

ここで、`Tweet`というクラスを導入している点に注意してください。`Tweetク`

ラスは、1つのツイートを表すデータを保持するPOJOです。POJOアプローチの利用は必須ではありませんが、アプリケーションの別の場所でこのTweetの概念を利用したい場合に、特定のクラスがあると便利です。Tweetクラスのコードはリスト5のようになります。

ObjectDataSourceBuilderの作成後、その流れるようなAPIを使用して、作成するObjectDataSourceに関する次のような指示を出しています。

- 解析後のエントリをTweetのインスタンスに変換すること
- 先ほど作成したDataSourceReaderからデータを取得すること
- 入力されたツイートはitemというXML要素内にあること
- 結果のエントリをallTweetsという既存のObservableListに追加すること
- 取得したエンティティをTableViewで表示する必要がある場合、ツイートの作成者とタイトルのみを使用し、公開日は使用しないこと

次に、ObjectDataSourceを作成します。データの取得は、DataSource.retrieve()メソッドを呼び出したときに初めて行われます。org.javafxdata.datasources.providerクラス内のすべての実装クラスは、retrieve()メソッドの呼び出し時にすぐに制御を返します。データの取得と解析は、JavaFX Serviceの概念を利用して、バックグラウンド・タスクで処理されます。そのため、retrieve()メソッドをJavaFXアプリケーション・スレッドから呼び出すことが重要です。

データを取得および解析して各エントリに分割する際に、解析後のエントリ

は適切なObservableListに追加されます。ObjectDataSourceBuilderで既存のObservableListを指定した場合は、その指定のObservableListが使用されます。ObservableListを指定しなかった場合は、初期段階では空の新しいObservableListが作成され、エントリの解析後すぐにデータが設定されます。その結果のObservableListは、ObjectDataSourceのgetData()メソッドでいつでも取得できます。DataSourceにはrunningPropertyも含まれています。runningPropertyは、DataSourceが現在データの処理中であることを示す便利なプロパティです。

なお、ObjectDataSourceのretrieve()メソッドの呼び出しによって、エントリの読取りと解析を実行中であるバックグラウンドのServiceオブジェクトが返されます。そのため、Serviceの既存のプロパティ(つまりstateProperty)を使用して、この処理の進捗状況を追跡し、問い合わせることも可能です。

ListViewとTableViewでは同じObjectDataSourceを使用できます。本記事のサンプルでは、ListViewを含むタブとTableViewを含むタブの2つを含む1つのTabPaneを示しています。

ListViewにデータを設定するコードは非常にシンプルです(リスト6参照)。allTweetsフィールドはObservableListインスタンスであり、このインスタンスのデータはObjectDataSourceが設定します。

TableViewのデータ設定と表示する列の定義は、リスト7のコードで行います。TableViewにallTweetsリストを設

LISTING 4

LISTING 5

```
ObjectDataSource createDataSource() {
    RestRequestBuilder rrb =
    new RestRequestBuilder("http://search.twitter.com")
        .path("search.rss")
        .queryParams("q", "javafx");
    DataSourceReader dataSourceReader = rrb.build();
    ObjectDataSourceBuilder<Tweet> builder =
    ObjectDataSourceBuilder.<Tweet>create();
    builder.itemClass(Tweet.class)
        .dataSourceReader(dataSourceReader)
        .itemTag("item")
        .resultList(allTweets)
        .columns("author", "title");
    ObjectDataSource dataSource = builder.build();
    return dataSource;
}
```

 [Download all listings in this issue as text](#)

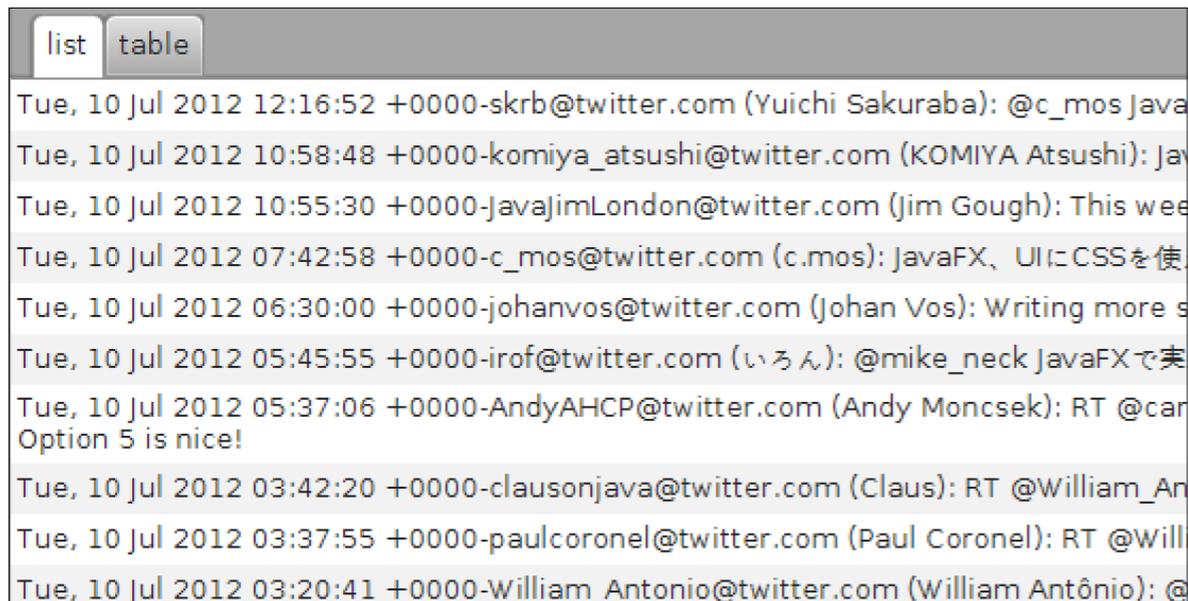


Figure 2

author	
skrb@twitter.com (Yuichi Sakuraba)	@c_mos JavaFXの
komiya_atsushi@twitter.com (KOMIYA Atsushi)	JavaFX はインスト
JavajimLondon@twitter.com (Jim Gough)	This weekend is a
c_mos@twitter.com (c.mos)	JavaFX、UIにCSSを
johanvos@twitter.com (Johan Vos)	Writing more sam
irof@twitter.com (いろん)	@mike_neck Java
AndyAHCP@twitter.com (Andy Moncsek)	RT @carldea: Re- Option 5 is nice!
clausonjava@twitter.com (Claus)	RT @William_Ant
paulcoronel@twitter.com (Paul Coronel)	RT @William_Ant
William Antonio@twitter.com (William Antônio)	@VladimirVivien

図3

定するほかに、データソースから返されたTableColumnインスタンスを列に設定しています。

リスト6とリスト7を組み合わせたコードの実行結果を図2と図3に示します。

セル・ファクトリ

JavaFXコントロールでは、コントロール自体の構成物(ListViewやTableViewのCellなど)と、コントロール内で表示されるコンテンツを明確に分離できます。コンテンツと、そのコンテンツのセル内での表現をつなぎ合わせるものがファク

LISTING 6 LISTING 7 LISTING 8

```
void buildListTab(Tab t) {
    ListView listView = new ListView(allTweets);
    t.setContent(listView);
}
```

Download all listings in this issue as text

トリです。

ListViewにはcellFactoryプロパティがあります。また、TableViewには、TableColumnインスタンスのListがありますが、このそれぞれのTableColumnインスタンスにもcellFactoryプロパティがあります。また、TableViewにはrowFactoryプロパティもあります。

まずは、ListViewについて確認します。リスト8のように、ListViewのcellFactoryPropertyを取得します。

Callbackインタフェースでは、次のように1つのメソッドを宣言しています。

```
public interface Callback<P,R>
    R call (P param)
```

そのため、カスタムのcellFactoryではCallbackの実装を行う必要があります。その実装には、ListView<T>パラメータを受け取り、ListCell<T>値を返す次のようなcallメソッドを追加します。

ListCell<T> call (ListView<T>)

TableView内のTableRowインスタンスを使用する場合も同様です。リスト9のように、TableViewのrowFactoryPropertyを取得します。

rowFactoryを指定したい場合は、次のようにcallメソッドを宣言したCallbackの実装を作成する必要があります。

TableRow<S> call (TableView<S>)

ただし、rowFactoryを使用するケースはほとんどありません。TableViewに対して推奨されるアプローチは、それぞれのTableColumnインスタンスにセル・ファクトリを導入することです。TableColumnは、その名前からわかるように、TableView内の1つの列のコンテンツを管理する役割を担います。

リスト10のように、それぞれのTableColumnには取得したcellFactoryプロパティが含まれます。



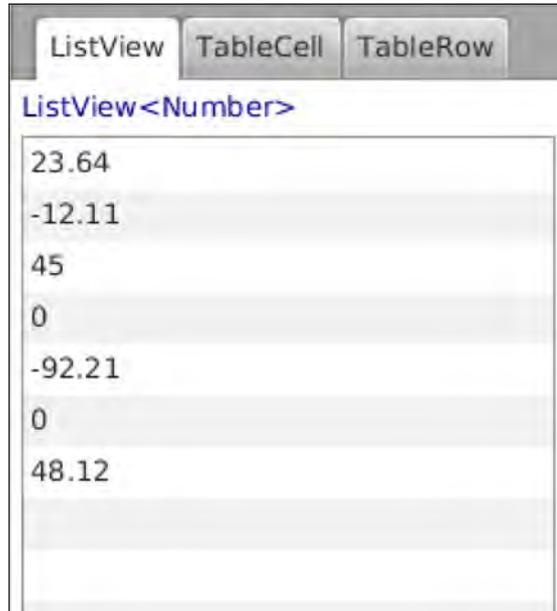


図4

表の特定の列に対して`cellFactory`のカスタム実装を指定する場合は、次のような`call`メソッドを定義した`Callback`クラスの実装を作成する必要があります。

```
TableCell<S,T> call
(TableColumn<S,T>)
```

これまでに説明したすべてのコードの実装は可能ではありますが、求める結果に対して必要となる定型コードの量が比較的多くなる傾向があります。

そのため、DataFXには多くの定義済みのセル・ファクトリが用意されています。現在JavaFX 2.2に含まれているセル・ファクトリの一部は、最初はDataFXで提供され、後でJavaFX 2リポジトリに移行されたものです。DataFXリポジトリには、まだJavaFX 2リポジトリに移行されていない、便利ではあるものの試行的なセル・ファクトリが含まれています。

セル・ファクトリの例: 例として、



図5

`MoneyCellFactory`を紹介します。まずはリスト11のコードを検討します。このシンプルな例では、`cellFactory`を指定せずに`ListView`を作成しています。その結果の`ListView`は図4のようになります。

図4のリストの数値を金額として表示する場合、`MoneyCellFactory`を利用できます。そのためには、リスト12の1行のコードを追加します。その出力は図5のようになります。

DataFXのすべてのセル・ファクトリと同様に、`MoneyCellFactory`にも、`ListView`インスタンスだけでなく`TableColumn`インスタンス用のセル・ファクトリを作成するための各種メソッドがあります。

正しいパラメータを正しい順序で指定しているかといった`Callback`インタフェースの実装について、開発者が心配する必要はありません。

LISTING 9 LISTING 10 LISTING 11 LISTING 12

```
public final
ObjectProperty<Callback<TableView<S>,TableRow<S>>>
rowFactoryProperty
```

Download all listings in this issue as text

まとめ

本記事では、高性能かつスレッド・セーフで柔軟なコードを使用して、効果的にデータを取得、解析し、さまざまなJavaFXコントロールに表示するためのDataFXの各種ツールについて説明しました。

DataFXは大きく分けて2つの相補的な要素で構成されています。

- データソース:データの取得と解析のツール
- セル・ファクトリ:多くのJavaFXコントロールにデータを容易に表示するツール

DataFXの各種ツールの導入方法を学ぶことで、JavaFXでの開発が少し楽になるでしょう。 </article>

LEARN MORE

- [DataFX](#)
- [The DataFX project](#)
- [DataFX source code repository](#)

GRAALの 将来

Oracle Labsの**Dr. Thomas Wuerthinger**、Graalプロジェクトの目標について語る

BY JANICE J. HEISS

PHOTOGRAPHY BY BOB ADLER

Oracle Labsが運営するGraalプロジェクトは、Java仮想マシン(JVM)のコンパイル時間やメモリ使用率を損なうことなく、卓越したコード品質を生み出す、Javaの動的コンパイラの実装を目指しています。結果として、アプリケーションと仮想マシン(VM)間のシームレスな統合が可能になります。このプロジェクトを率いるのは、Oracle Labsの技術スタッフのシニア・メンバー、Thomas Wuerthingerです。Wuerthingerは、2011年にオーストリア、リンツのヨハネス・ケプラー大学でコンピュータ・サイエンスの博士号を取得しました。以前は、IdealGraphVisualizer、Crankshaft/V8最適化コンパイラ、Dynamic Code Evolution VMの開発に取り組んでいました。Graalの開発状況について話を聞きました。[Graal Project](#)

Java Magazine: [Graalプロジェクト](#)が実現しようとしている



ものは何ですか。

Wuerthinger: 簡単に言うと、このプロジェクトは、“独自のJを利用するJVMの探求”です。私たちは、主要なコンポーネントがJavaで記述されているVMを作成したいと考えています。そうすることで、それらのコンポーネントはJavaの安全な実行モデルおよび

ツールリングから恩恵を受けられるのです。また、Graal VMのモジュール設計により、拡張性と保守性の向上を目指しています。

Java Magazine: 目標達成まで、どのくらいのところに来ていますか。

Wuerthinger: これまでは、プロジェクトの重点はGraal VMのインタフェースの開発にありました。拡張が簡単になるからです。これによって、VMを試しやすくなり、私たちのビジョンにも貢献することになります。

さらに、HotSpot VMのクライアントとサーバー構成間にあるピークパフォーマンスを提供するJavaの動的コンパイラのプロトタイプを開発しました。Jikes RVM (Research Virtual Machine)やMaxine VMのように、私たちのシステムは、Java言語を使用したシステム・ソフトウェアの記述が可能であることを示しています。

Java Magazine: Graalコンパイラによって、現状では不可能な方法でJavaライブラリの機能を拡張できるため、さらに多くの新しい言語を効率的に実装できるという議論があります。これについて、詳しくお聞かせください。たとえば、どのような新言語が実装されるのでしょうか。

すべてをサポート

“複数の言語をサポートすることが、Graalのおもな目標です。あらゆる言語に、Javaプラットフォームの機能を提供したいのです”



オラクル本社でのミーティングの合間に電子メールをチェックする Dr. Thomas Wuerthinger

Wuerthinger: 複数の言語をサポートすることが、Graalのおもな目標です。あらゆる言語に、Javaプラットフォームの機能を提供したいのです。特に、私たちは、言語の実装者がコンパイラやVMの詳細を心配することなく、対象言語のセマンティクスの実装に集中できるような複数言語フレームワークを開発しました。言語のセマンティクスは、その言語向けのAST(抽象構文ツリー)インタプリタを実装するJavaプログラムによって指定されます。私たちは、そのフレームワークに実装された言語に対し、Graalコンパイラで高パフォーマンスを実現できるようにする技術を開発しました。APIを成形するため、いくつかの言語向けのプロトタイプを開発し、言語の実装者を招いてこのフレームワークを試してもらっています。

Java Magazine: なぜ一般的なJava開発者は、数年先を見越してGraalを気にかける必要があるのでしょうか。今できないことの中で、何ができるようになるのでしょうか。

Wuerthinger: Graalプロジェクトは、ほかの多くのOpenJDKプロジェクトに比べて、経験に基づく部分が大きいのです。そのため、私たちがしていることが最終的に製品として出荷されると約束することはできません。それでも、私たちはGraalプロジェクトには積極的に貢献し、今後数年間にわたってJavaエコシステムに活力をもたらすと信じています。特に私たちが期待するのは、GraalによってJVM内のJava言語の利用方法についてより詳しく知ることができるということです。これによって、Java自体のパフォーマンスが向上し、追加の言語のサポートもより良いものに



Wuerthinger(右から3番目)と、
オーストリア、リンツのヨハネス・
ケプラー大学でGraalプロジェクトに
貢献している博士課程の学生たち

写真提供:ヨハネス・ケプラー大学

なります。

また、Graalの柔軟性により、クラウド・インフラストラクチャ、異機種コンピューティング環境、組み込みデバイスなどのさまざまなドメインにおけるアプリケーションの実現が可能になります。私たちはJavaの「Write once, run everywhere. (1回書けば、どこでも実行)」というスローガンを守り続けたいと考えています。

Java Magazine: Graalに興味がある開発者は何をすべきですか。

Wuerthinger: Graalの開発に参加するためのハードルは、できる限り低くしたいと考えています。そのため、プロジェクトのトップページに、Graalの構築方法と実行方法に関する説明を少し載せました。また、開発者の皆さんには、私たちの[OpenJDKメーリングリスト](#)に参加するよう呼びかけています。週に一度、Graalコードベースへの変更がそのリストに配信されます。

Java Magazine: Graalプロジェクトでは、目標の1つとして、VM内でのJavaの利用を挙げています。JVMの一部をJavaで記述することに関連する特有のメリットや課題は何ですか。

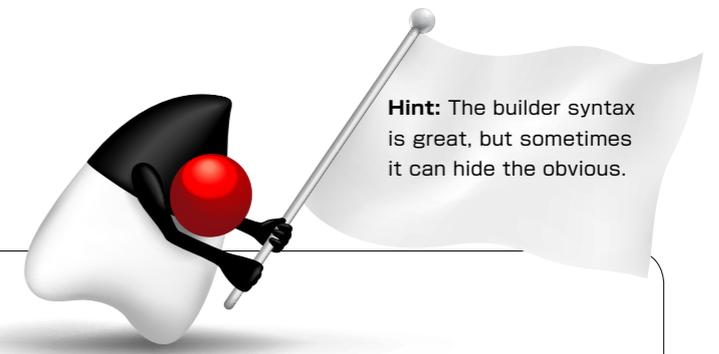
Wuerthinger: おもな利点は、コンパイラを管理された環境で実行することによる堅牢性の向上です。これは、たとえばコンパイラにエラーがあった場合、VMプロセスがクラッシュするのではなく、ヌルポインタ例外が生じることを意味します。また、Javaリフレクションおよびアノテーションも使用できます。コンパイラとアプリケーションの間に言語の障壁がないため、Javaアプリケーションによって簡単にコンパイラに影響を与えて拡張することができます。2つの分野に課題があります。コンパイラはメモリ効率の良い状態にし、ガベージコレクションのパフォーマンスを低下

させないようにする必要があります。また、起動を迅速にするため、コンパイラを事前コンパイルしておく必要があります。

Java Magazine: Graalへの取組みは、何が魅力ですか。

Wuerthinger: 私は、Javaでのプログラミングのあらゆるツールや利点を維持しながら、システム・ソフトウェアで作業するという点を気に入っています。また、JVM上で自由闊達な経験ができることを楽しんでいます。経験に基づくというプロジェクトの性質のため、私たちはすぐに現実的な障害に阻まれることなく、新しい方法を試すことができます。さらに、経験のある開発者や、修士課程や博士課程の好奇心旺盛な学生たちが入り混じったGraalチームで働くことに魅力を感じています。 </article>

Janice J. Heiss: オラクルのJava編集者であり、Java Magazineのテクノロジー編集者



Hint: The builder syntax is great, but sometimes it can hide the obvious.



2012年7/8月号で、Jason HunterとBoris Shukhatが接続プール・コードの課題を出しました。その課題とは、プール・クラスの一部のコードが示され、大幅に再設計することなくそのコードを修正するというものです。

正解は4番です。ConnectionPool.javaコードでは、hashCode()とequals()はオーバーライドされないという誤った仮定をしていました。このため、新しいドライバへアップグレードすることで、プログラムが壊れてしまったのです。新しい実装では、デフォルトの参照型等価の代わりに、オブジェクト等価を使用してequals()が実装されたためです。4番の方法では、プログラムをhashCode()およびequals()の実装から完全に独立させることができます。解決策は、実際のConnection実装をラップし、内部でコールをルーティングするConnectionインタフェースの実装を安全に作成することです。それによって、プログラムは常にオブジェクトの信頼性の高いhashCode()およびequals()メソッドを使用することになるでしょう。

今回は、Stephen Chinからの問題です。Chinは、オラクルのJavaエバンジェリストであり、『Pro JavaFX 2』(Apress, 2012年)の共著者です。

1 問題

JavaFX Media APIにより、ビデオ・コンテンツをとっても簡単にデスクトップ・アプリケーションに追加できます。APIが非常に豊富な機能を備えている一方、多くのクラスが関連するため、注意しないとシンプルなユースケースでも間違いを犯してしまう場合があります。

2 コード

以下のMediaQuizクラスのApplicationのstartメソッドのコードで、ビデオ・クリップが再生されないのはなぜですか。

```
stage.setScene(SceneBuilder.create()
    .width(960).height(540)
    .root(
        StackPaneBuilder.create().children(
            new MediaView(
                new MediaPlayer(
                    new Media(
                        getClass().getResource("quiz.mp4").toString()
                    )
                )
            )
        )
    )
    .build());
stage.show();
```

以下のソース・レイアウトを使用しました。

- org
 └─ steveonjava
 └─ MediaQuiz.java
 └─ quiz.mp4

3 正しいのはどれ?

- 1) メディア・ファイルをロードするためのリソース・パスが間違っている
- 2) JavaFX 2は、H.264でエンコードしたビデオをサポートしない。
- 3) ビデオを有効化するための制御機能がない
- 4) 正しいStackPaneレイアウト制約セットがなく、MediaViewが表示されない

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