

JavaTM magazine

By and for the Java community



JAVA GROWS UP }

05 MARK REINHOLDとJOHN ROSEへのインタビュー

22 IOT: JAVAで
コーヒーを淹れる

15 JPAの
最新情報

51 ロボット: マシン
内部のJAVA

ORACLE.COM/JAVAMAGAZINE

//MAY/JUNE 2015 /

```
List<Integer> transactionsIds =  
    transactions.stream()  
        .filter(t -> t.getType() == Transaction.GROCERY)  
        .sorted(comparing(Transaction::getValue).reversed())  
        .map(Transaction::getId)  
        .collect(toList());
```

// as a parallel stream:

```
List<Integer> transactionsIds =  
    transactions.parallelStream()  
        .filter(t -> t.getType() == Transaction.GROCERY)  
        .sorted(comparing(Transaction::getValue).reversed())  
        .map(Transaction::getId)  
        .collect(toList());
```

```
List<Integer> transactionsIds =  
    transactions.stream()  
        .filter(t -> t.getType() == Transaction.GROCERY)  
        .sorted(comparing(Transaction::getValue).reversed())  
        .map(Transaction::getId)  
        .collect(toList());
```

// as a parallel stream:

```
List<Integer> transactionsIds =
```

言語の進化と JVM の
イノベーションの
20 年

ORACLE®



06

11

02

04

15

22

29

34

38

41

46

51

01

A middle-aged man with short, graying hair is walking towards the camera on a city sidewalk. He is wearing a light blue long-sleeved button-down shirt, a brown leather belt, and blue jeans. He holds a black folder or tablet under his left arm. The background is a blurred city street with other pedestrians and buildings.

Java Magazineは新しい方向へ

が共有するプログラミングへの愛情、優れたツールやテクノロジーを求める情熱、そして誰もがJavaやJVM言語から得られる不朽の楽しさを映し出し、それらに応えるものにしていきます。

そのために、もっと多くのJava専門家、熟練のプログラマー、アルゴリズム・マニア、設計の達人、ビルドの天才、劣悪なコードの指摘者、経験豊かな講師、スキルの豊富なアーキテクトに、Java Magazineのページ上で知恵を共有してもらうよう協力を求めています。ツール類、特にオープンソース・ツールと、重要なオープンソース・プロジェクトについても探究します。

また、プログラミングとあまり関連のない部分を中心に、一部の記事を取りやめる予定です。Java Magazine では、過去の事例についてはあまり取り上げず、有益な事例については関連する技術トピックで直接紹介するようにします。他にもまだあります。たとえば、書籍の推薦コーナーでは、詳しいレビューをレビューアーの署

新しい仕事の喜びとともに、私が目指す終着地までの長い道のりも感じています。Java Magazineの新しい方向性に関する計画は次のとおりです。

簡単に言えば、Java Magazineを根本的にもっと技術的な方向へ進めようと思っています。私は編集チームとともに、日常的にコーディングに携わるJava開発者と真っ向から向き合います。Java Magazineを、開発者



ORACLE.COM/JAVAMAGAZINE //////////////////////////////////// MAY/JUNE 2015

簡単ですね。このように日付と時間の新APIを使えばアルゴリズムのデバッグに力を注ぐことができます。日付フィールドが0、1の順なのか、また1900から始まるのか、頭を悩ませる必要はありません(この数値の意味は、Chinの[ZombieTimeプレゼンテーション](#)のスライド10で確認してください)。

JOHN ROSEに 聞く

今後もJVMにはProject ValhallaとProject Panamaによって
便利な機能が追加される**TIMOTHY BENEKE**

Javaへの賛辞を送るには、JVMの並外れた万能性について語らなければなりません。今やJVMは、Clojure、Groovy、JRuby、Jython、Kotlin、Scalaなど、非常に多くの言語をホストしています。JVMアーキテクト、John RoseのJavaに関する経歴は、Sun Microsystemsで働いていた1995年まで遡ります。Roseは、Java インナー・クラスの設計、SPARCへのJava HotSpot VM初回移植、Unsafe APIの開発に従事してきました。彼の功績によって、HotSpotやOpenJDKスタックのパフォーマンスが向上しました。このパフォーマンス向上はハードウェア・アーキテクチャからコード・ジェネレータ、ライブラリ、プログラミング言語まで、あらゆる領域に波及するものです。Roseは現在、OpenJDKの取組みの1つのDa Vinci Machine Projectのリード・エンジニアであり、さらにJSR 292のスペックリードも務めています。JSR 292は、動的呼出しやその関係機能（型プロファイル、改良版のコンパイラ最適化など）をJVM標準で新たにサポートするための仕様を定義しています。そんな彼に、JVMの過去と未来に関する見解を話してもらいました。



Rose関わった最初のJVMプロジェクトは、Java 1.1のインナー・クラスの設計と実装だった。

Rose:若い頃はLispプログラミングを楽しんでいたため、「Lisp難民」と自己紹介することもあります。Lispはガベージ・コレクション、安全な動的性質、そしてリッチ・ライブラリでの安全な型付けの機能を備えるマネージド言語です。多くの言語ゲークと同じように、私も独自のLisp VMを開発し、Sunのいくつかのプロジェクトでこの独自のVMを活用しました。

覚えています。そして、Javaに関するある問題を抱えていたJames Gosling氏に助けを求められました。それは、メソッドのポインタや委譲の方法に関することでした。結局私はJava 1.1のインナー・クラスの設計と実装を行いました。つまり、かなり初期の段階からJavaに関わっています。

Java Magazine:今でも自らコーディングしていますか。

Javaは今後も当然のように成長し、現在C++で行われているタスクに、より多く対応できるようになる見込みです。C++を使ってイライラするときは、たいてい、間違いを犯せないJVMの機能を開発しているときです。チーム・メンバーはC++の苦勞を互いに分かち合い、他のみんながVMでJavaを使っているのになぜか自分たちはC++で開発しなければならないことを慰め合っています。そして、他の大多数の開発者をC++から解放するために自分たちがC++でコーディングしているのだとよく言っています。



Java Magazine: JVMの歴史的側面について話してもらえますか。

Rose:元々のJavaの考案者は、一度にすべての問題を強引に解決できないことを理解していたので、階層的に取り組むことにしました。彼らはJavaだけでなく、仮想マシンも生み出しました。特に重要なのが、Javaコンパイラのコンパイル先となるバイトコードとオブジェクト・アーキテクチャです。Java開発者にとっては、これらは「Javaの」バイトコードですが、実際はそれをはるかに凌ぐ意味があります。バイトコードには独自のアイデンティティと独自の特別な能力が備わっており、バイトコードそのものが、非常に強力に洗練された設計に基づく実行モデルだと見なせます。

初版のJVMマニュアルには、バイトコードが単なるJavaの複製ではなく、他言語もサポート可能であることが明記されていました。そして、アーキテクチャ・マニュアルからヒントを得た開発者や、元々そのような目的を果たす上でバイトコードが非常に魅力的であると感じていた開発者が、JVMに対応した他言語の開発を始めていました。そうして、Javaの誕生からわずか数年後には、JVM上で実行される言語実装が文字通り何百も知られるようになりました。はじめの頃は、KawaのようなさまざまなScheme系言語が登場しましたが、今でもJavaバイトコードで実行される新しい言語が続々と生み出されています。このような状況には心が躍ります。

Java Magazine: JVMはどのように開発され、時代とともにどのように変化してきたのでしょうか。

Rose: Sun Microsystemsは2005年頃、動的言語をサポートするためにJVMへ新しい呼出しモードを追加しようと真剣に取り組んでいました。それまでの呼出しモードはすべて静的型付け言語、特にJavaに対応するように構成されていましたが、動的言語や動的なセマンティックをもっと直接的にサポートする新しい呼出しモードや手法を追加したかったのです。この機能は、詳細の検討前から「invokedynamic」命令という名前と呼ばれていました。

しかし、その実装には5年もかかりました。動的呼出しの方法

について多くのアイデアがあり、設計がバラバラの方向に進んだからです。最初のプロトタイプの方角性は誤っていて、JSR 292の専門家グループ[編集注:Java Community Processの専門家グループのこと]に指摘を受けました。そのため設計が見直され、導入はJDK 7になりました。それ以降、invokedynamicはJavaスタックの重要な部分を占めています。invokedynamicは、まるで万能ナイフのように汎用性の高い呼出しであり、動的メソッド呼出しの実装だけでなく、ある種の静的呼出しの実装にも使えます。これは以前の命令では不可能でした。まさに要点を押さえた命令であり、JVMの機能全体にアクセス可能で、JVM上で言語の実装を行う上でも十分な柔軟性が提供されます。私たちは概念実証のため、JRuby開発リーダーのCharlie Nutter氏から早い

段階で協力を得ました。そして、`invokedynamic`を使ったJRubyのプロトタイプ実装を複数、素早く開発してもらいました。このように、本当に早い段階から動的言語の開発者に協力を仰ぎました。

最近では、Oracleがまったく新しいJavaScript実装としてNashornを開発しまし

た[編集注:NashornはJDK 8に搭載されている]。Nashornは、JVM上で実行される高性能JavaScriptエンジンですが、当初から **invokedynamic** に基づいて設計されています。

次のことははっきりさせておきたいと思います。バイトコード・アーキテクチャに変更を加える際には、将来にまで影響するため、検討を重ね、最終的に良い設計に行き着くまでに何年もかかることもあります。そして、いざ実装を決定したら、設計最適化の原動力を確立するためにさらに数年かかります。JVMがゆっくりと進化しているのはこのためです。先日開催されたJavaOneでのJames Gosling氏のことばを借りれば、「Java、そしてJVMでうまく行くと分かるまでは実行に移しません」。

Java Magazine: JVMはどこを目指していますか。

Rose: オラクルおよびJavaコミュニティは、ValhallaとPanamaという2つのOpenJDKプロジェクトに大きな投資を行っています。

JAVA VS. C++

「チーム・メンバーはC++の苦勞を分かち合い、他のみんながVMでJavaを使っているのになぜか自分たちはC++で開発しなければならないことを慰め合っています」

Project Valhallaは最終的に、参照とプリミティブの差を埋めて、ユーザーが新しい型をプリミティブのように定義できることを目指しています。この新しい型は「値型」と呼ばれます。Javaには複素数型や符号なし整数型、128ビット整数型がありません。しかし、値型があれば、これらの型をVM固有の型ではなくライブラリのコードとして作成できるようになります。これはすばらしい構想です。値型が必要になる基本的な理由ですが、値型は物理的に、よりフラットでコンパクトなデータ構造を自然に保持する単位となるからです。現在のJavaアーキテクチャでも、「値型がなく、新しいプリミティブを作成できないとしても、クラスを使って128ビット整数や符号なし整数、複素数を作成できる」とは言えますし、開発者は実際にこのように対応しています。しかしこの対応では、Javaネイティブのオブジェクト指向データ・モデルに縛られ、あらゆる場所にポインタが置かれるという問題が生まれます。ポインタで到達可能なオブジェクトを作成してしまうと、そのポインタの除去は困難です。いわゆる「アイデンティティ・セマンティック」の影響で、コンパイラにとってポインタを無視することは困難、あるいは不可能なのです。これが仮に問題にならないなら、Javaにはプリミティブ型は不要で、すべてがオブジェクトで表されたはずで、値型を使えば、intと同じようなパフォーマンス感覚で、複素数を扱うことができま

す。整数データに対して `Java.lang.Integer` を使うのは躊躇します。それと同じ理由で、Javaの複素数クラスの利用もためられます。しかし、複素数がオブジェクト型ではなく値型になれば、間接的な参照はなくなりフラットなデータ表現になるため、パフォーマンスの計測ではintに近い結果が得られます。

これは、「クラスのようにコーディングし、intのように動く」という値型のスローガンそのものです。一見クラスに見える

複素数をライブラリ構造内で定義しますが、そのデータはintのような感覚で扱えます。

値型の追加によって、他言語からJVMへの移行も簡単になります。Javaにはないデータ型を持つ言語が、データ型をJVMに合わせて変更する必要がなくなります。元の言語のデータ型を反映した新しいデータ型を作成できるからです。これは今でも可能です。

が、ポインタや記憶域の割当てにコストがかかります。

Java Magazine:Project Panamaについて教えてください。

Rose: Project Panamaでは、ネイティブ・コードとのより堅牢な相互接続性を確立しようとしています。開発者がJavaを好む理由は、記述しやすい言語だからです。しかし、C APIなら低レベル・システム・ライブラリが使えるなどの理由で、CやC++のプログラミングが必要になることもあります。そうすると、CのコーディングのためにCエディタを取り出し、ヘッダー・ファイルをインクルードしなくてはなりません。さらに、完成したコードをJavaプロジェクトに組み込むためには、大量のJava Native Interface[JNI]コードを記述する必要もあります。しかし、JNIは意図的に使いづらいものになっています。Javaが誕生した頃は、独立性を確保するため、ある程度の全体性を保持しながら自身の実行制御を行う必要がありました。そのため、JVMが進んでネイティブ・ライブラリと信頼性を共有することはありませんでした。ネイティブ・ライブラリと連携する場合、必要なJNIバインディングの開発作業に数か月かかることもありました。

Cの世界からJVMの世界に移るには、Cを置き去りにするか、多大なコストを払ってCを持ち込むしかなかったのです。まるで輸入関税のようです。Javaは今、コンピューティング分野で安定した地位を得ています。そして、Cの世界からJavaの世界に関税なしで入ることができれば、それはJavaにとって有益なはずで、これを実現するのがProject Panamaです。

イメージはこうです。Javaが太平洋でCが大西洋だとすると、Panamaはこの2つの海洋をつなぐ橋となり、運河となります。私はブログに数か月前、「[The Isthmus in the VM](#)」という意見を投稿し、JavaとCを接続するリスクと利点について述べました。今は幸いにも、Project Panamaは順調に進行しています。JVMユーザーはいつの日か、今よりもさらに強力なデータ型を備える幅広い言語を利用できるばかりか、オーバーヘッドをはるかに抑えた手法でCライブラリを利用できるようになるはずです。

Java Magazine: バイトコード変換ツールについてはどういう状況でしょうか。

Rose: まずはその背景について説明しましょう。柔軟な仮想マシンを高速で実行するという考え方は、かなり昔からあるものです。

JVM言語

「Clojureは、マッド・サイエンティストたちが並行プログラミングと永続的データ構造をLispのような独特な言語設定で再検討したライブラリです」

MARK REINHOLDに聞く

Javaの幅広い利用を促した変化とこれからの変化について
BY TIMOTHY BENEKE

Mark Reinholdは、オラクルのJava Platform Group担当チーフ・アーキテクトであり、Sun Microsystemsからオラクルへとわたった19年の間、Javaプラットフォームへの数々の貢献を果たしてきました。文字ストリームのリーダー/ライター、NIO（高パフォーマンスI/O API）、ライブラリのジェネリクス対応、サービス・ローダーなどを担当し、Java 1.2リリースとJava 5.0リリースではリード・エンジニア、Java SE 6ではスペックリードを務めました。また、JDK 7（Java SE 7）とJDK 8（Java SE 8）ではプロジェクト・リード兼スペックリードという立場でした。現在はJDK 9プロジェクト、およびOpenJDKコミュニティでJigsawプロジェクトのリーダーを務めています。OpenJDKコミュニティでは運営委員会のメンバーでもあります。

かつてのLispハッカーが変装していると自称するReinholdは、マサチューセッツ工科大学で計算機科学の博士号を取得しました。大学では、ガベージコレクション、コンパイル手法、型システム、セマンティック、仮想化、プログラムのパフォーマンス分析について研究しました。

JAVAを利用すること

「内部的には複雑であっても、Javaには敷居の低さがあるのです。Javaには高い『DWIM (Do What I Mean)』の要素があります」



Java Magazine: Javaの長年の歴史でもっとも驚いたことは何でしたか。

Reinhold: 基盤の強さだと言わざるをえません。James Gosling氏を含むJavaの初代開発チームが、いかに強固な基盤をJava仮想マシン[JVM]とJava言語の仕様に組み込んでいるかを、当初は誰も理解していなかったと思います。Javaの基盤は明快で、しかも、あえて変更する気にならないように、自己完結性や相互依存性もそれほど高くなく、進化可能な設計でもあります。

たとえば、VMの仕様を調べると、VMの構造、クラス・ファイルの定義方法、バイトコード全体の構成とその意味について非常に明快に記されています。それでも、未使用のバイトコードが多く存在するため成長の余地があります。また、バイトコードが存在する概念フレームワークにも、柔軟性の面で可能性があります。Javaの言語仕様にも同じような特徴があります。

これまで何度もJavaが死亡宣告されてきたのも驚きです。死亡宣告はまだ実現しそうにないのですが。

Java Magazine: 可読性、単純さ、普遍性など、James Gosling氏の言う「Javaの感覚」を持続することが重要だと常々言っていますね。

Reinhold: それらの特性は本当に重要です。私たちはその価値を早くから認識し、それらを守るために一生懸命頑張ってきました。その中でも可読性はもっとも重要です。ソフトウェアの場合、開発者が本番用のコードの開発にかける時間は、そのコードの維持、バグ修正、拡張のために、その開発者や後を引き継いだ担当者がコードを読む時間と比べればごくわずかです。

単純さとは、プラットフォームの理解しやすさのことです。本番品質の言語やそれを取り巻くプラットフォームは、最新のコンピューティング・システムに内在する複雑さに対応するために、ある程度内部的に複雑である必要があります。

しかし、内部的には複雑であっても、Javaには敷居の低さがあります。Javaには高い「DWIM」要素があります。すなわち「Do What I Mean」（思っているとおりに動作する）ことです。Javaの初心者でも、使いやすいIDEで基本構造を学べば、本当に簡単にやっていけます。VM仕様を

読んだり、言語仕様を覚える必要はありません。これらの参照資料は簡単に入手でき、最終的には必要になるかもしれませんが、ただ「思っているとおりを書く」だけで、その開発中に意図したことの大部分ができます。Javaプログラマーとして現場に臨み、問題解決にも取り組みます。言語やプラットフォームの複雑さはJavaプログラマーの視界にはあまり入りません。

別の言語スタイルを好む開発者は反対しますが、こうしたスタイルの言語としては、Javaはかなり成功しています。

Java Magazine:なぜキャリアの大部分をJavaに注いできたのでしょうか。

Reinhold:私を最初にJavaに引き付けたものが、今でも私を留まらせています。プログラミング言語に深い関心を抱く人間にとって、何百万もの開発者が毎日利用するものに関わる機会というのはほとんどありません。本当に面白くやりがいがあり、理屈では語れない、ある種のスリルがあります。約20年前にJavaを始めた頃と同じくらい、今でもそのスリルを楽しんでいます。

Java Magazine:1日の仕事について少し教えてください。今でもコーディングはしていますか。

Reinhold:今でもコーディングしていますし、JDK 1.1から始まるすべてのリリースでコーディングをしてきました。あまりできなかったときもありますけどね。今は、若いエンジニアのメンタリングや開発者へのアドバイスにかなりの時間を注いでいます。古参のひとりとして人の知らないような事実も数多く知っているので、Javaプラットフォームの歴史を語る役割も担うようになりました。また、設計作業もかなり行っています。私は現在、Java SE 9に向けて開発中のモジュール・システムのスペックリードを担当しており、そこでもコーディングや設計、アーキテクチャ、言語の変更を行っています。

Java Magazine: オラクルがSun Microsystemsを買収してから5年以上が経ちました。オラクルによるJavaの管理についてはどう感じていますか。

Reinhold: かなりいいのではないかと感じています。2つの大企業の合併という最初の難関を突破した後、Javaの活気を維持することを最優先したオラクルの決定は賢明なものでした。この5年間での投資やリリース状況を見れば、オラクルが真剣にその約束を果たしてきたこと

明快なコード

「可読性はもっとも重要です。ソフトウェアの場合、開発者が本番用のコードの開発にかかる時間は、その開発者や後を引き継いだ担当者がコードを読む時間と比べればごくわずかです」



JOSH JUNEAU

Josh Juneau:
アプリケーション開発者、システム・アナリスト、DBA。開発ではおもにJavaとその他のJava仮想マシン(JVM)言語を利用。Oracle Technology NetworkおよびJava Magazineの技術記事を執筆している。『Java EE 7 Recipes』、『Introducing Java EE 7』(共にApress、2013年)、『Java 8 Recipes』(Apress、2014年)の著者。

JPAの新機能

属性変換、スキーマ生成、拡張SQL問合せなど、多くの改善点が目玉となるJPA 2.1

Java Persistence API (JPA) は、リレーショナル・データベース主導型のJava EEアプリケーションに欠かせない要素です。JPA 2.1はマイナー・リリースではありますが、開発者の総合的な生産性の大幅な向上につながる強力な新機能が追加されています。

本記事では、Java EE 7で導入されたJPA 2.1の最新機能をいくつか説明します。これらの新機能を現在のJava EEアプリケーションに取り入れることができるように、実際のアプリケーションでの使用例を取り上げ、JPA 2.1の新機能について深く掘り下げていきます。

本記事では、例としてAcmePoolsアプリケーションを使用します。これはJava EE 7で開発されており、フロント・エンドにJavaServer Faces (JSF)、データ処理にEnterprise JavaBeans (EJB) Beanを使用しています。このアプリケーションは、元はOracle Technology Networkに掲載されている「PrimeFaces in the

Enterprise」という記事で作成したものです。

データベース・スキーマと型マッピング

データベースのテーブルをJavaオブジェクトでモデル化する場合、開発者はいくつかの難題に直面することになります。JPA 2.1では、データベース・スキーマと型マッピングに関するさまざまな拡張が行われており、リレーショナル・データベースとJavaオブジェクトの間の移行が容易になっています。本セクションでは、こうした移行に役立つ2つの新機能である、属性変換とスキーマ生成について説明します。

属性変換: データベースのテーブルをオブジェクトとして表現するためには、それぞれのデータベース型をJavaの型にマッピングしなければなりません。これは、型マッピングと呼ばれています。JPAでは、Javaの型とデータベースの型との変換が自動的に行われない場合があります。一例として、JPAに用

意されているマッピングが、アプリケーションで必要とされるマッピングと異なる場合があります。JPA 2.1では、そのような場合に属性コンバータを記述して、どのデータベース型をどのJavaの型に対応させるかを指定できます。

Hibernateのカスタム型と同様に、データベース属性(列)の型とJavaオブジェクト型の変換を行うために、基本属性またはコレクション型の要素に属性変換を適用できます。たとえば、サンプル・アプリケーションでは、JavaのBoolean型を表すために、データベースで0と1ではなく、NとYの文字列値を使用したいと考えました。そのために、Javaのfalse値をNに、Javaのtrue値をYに変換するコンバータを記述して、変換された値をVARCHARフィールドに格納

します。これを実現するためには、`javax.persistence.AttributeConverter`インタフェースを実装するコンバータのクラスを作成する必要があります。このインタフェースは値`X`と`Y`を引数に取ります。`X`はJavaの型、`Y`は対応するデータベース型です。

アプリケーションの移植性を保つため、型コンバータは、**Id**属性、**version**属性、**relationship**属性や、**Enumerated**または**Temporal**と指定された属性には適用すべきではありません。**リスト1**に、BooleanをStringに変換するコンバータの例を示します。

コンバータは自動的に適用することもできますが、[@Convert](#)アノテーションでコンバータのクラス名を指定して適用することもできます。サンプル・アプリケーションでは、

スキーマ

JPA 2.1は、アノテーションの組合せとプロパティの指定によって自動スキーマ生成を実現しています。

リスト2のように、永続化する際や取得する際にコンバータを使用するエンティティ・クラスの属性に@Convertアノテーションを指定しています。

属性変換によって、エンティティ・クラスに余計な変換ロジックを組み込むことなく、簡単にデータベース・フィールドの型とJavaの型のカスタム・マッピングを設定できます。

スキーマ生成:かつては、あるデータベース向けのエンティティ・クラスを作成する場合、手動での生成か、IDEやプロバイダ固有の実装などを使用した非標準的な方法による自動生成が行われていました。エンティティ・クラスの手動生成は面倒でエラーも発生しやすく、IDEやプロバイダ固有の実装もベンダーによる囲い込みにつながります。JPA 2.1で自動スキーマ生成が導入されたことで、こういった問題が解決されました。

JPA 2.1は、アノテーションの組合せと、アプリケーションの永続コンテキスト内でのプロパティの指定によって自動スキーマ生成を実現しています。

スキーマの生成に対応するためには、[persistence.xml](#)ファイルにスキーマの生成方法を指定するプロパティを追加します。データベース・スキーマの生成のために指定することができるプロパティのうち、使用頻度が高いものを**リスト3**に示します。すべてのプロパティの

リストは、[Java EE 7チュートリアル](#)を参照してください。

リスト4に、JPAのスキーマ生成プロパティを含む`persistence.xml`ファイルの例を示します。スクリプト・ソースを指定するプロパティはSQLスクリプトを指しており、そのスクリプトには、データベース・オブジェクトの作成、ロード、削除に必要なSQLコードが記述されています。また、プログラムでデータベース・スキーマを生成するためのAPIも提供されています。通常、上記のスクリプトはアプリケーションの`META-INF`フォルダに格納します。

JPAのスキーマ生成がほとんどのIDEやプロバイダ固有の実装よりも優れているもう1つの点は、索引と外部キーをサポートしている点です。索引は@Indexアノテーション、外部キーは@ForeignKeyアノテーションで指定します。データベース属性に索引を指定するためには、@Tableアノテーションにindexes属性を指定し、その中に@Indexアノテーションを記述してそれぞれの索引を指定します。

リスト5に、Customer エンティティに索引を指定する方法を示します。
@ForeignKey アノテーションは、エンティティの属性に指定する @JoinColumn アノテーションの内部に記述します (**リスト6**)。
@ForeignKey アノテーション


リスト1 / リスト2 / リスト3 / リスト4 / リスト5 / リスト6

```
@Converter
public class BooleanToCharacterConverter implements
    AttributeConverter<Boolean, String> {

    @Override
    public String convertToDatabaseColumn(Boolean x) {

        if(x){
            return "Y";
        } else {
            return "N";
        }
    }

    @Override
    public Boolean convertToEntityAttribute(String y) {
        String val = (String) y;
        if(val.equals("Y")){
            return true;
        } else {
            return false;
        }
    }
}
```

 すべてのリストのテキストをダウンロード

ンが指定されると、永続性プロバイダによるデフォルトの設定がオーバーライドされることに注意してください。

データベースの構成要素の呼出し

JPA 2.1では、データベースの構成要素の呼出しに新しい技術が導入され、パラメータの指定と値の取得が可能になりました。その結果、今まで

以上に簡単にデータベースのプロシージャや関数を呼び出せるようになり、エラーも起こりにくくなっています。本セクションでは、この新機能の中から2つを説明します。

名前付きストアド・プロシージャ:場
合によっては、アプリケーションからデータベース内のストアド・プロ
シージャを呼び出すことが必要になり
ます。かつてはネイティブSQL呼

JPA 2.1

JPA 2.1リリースで導入されたアノテーションは、データベース内に格納されているプロシージャを呼び出す便利な方法です。

出しで対応していましたが、JPA 2.1 リリースでは、データベース内に格納されているプロシージャを呼び出す便利で標準的な方法として、`@NamedStoredProcedureQuery`アノテーションが導入されました。このアノテーションを使用すると、ストアド・プロシージャを識別子にマッピングし、`resultSet`を1つまたは複数のエンティティ・クラスにマッピングすることが可能になります。

この機能を説明するために、指定されたCustomer IDに基づいてプール情報を取得するAcmePoolsデータベース内のストアド・プロシージャを呼び出すことを考えてみましょう。このストアド・プロシージャは、**リスト7**に示すようなものです。エンティティ内でストアド・プロシージャを宣言するためには、@NamedStoredProcedureアノテーションを指定し、その中にname、resultClass、procedureName、parametersを記述します。

リスト8に示す例では、OBTAIN_CUSTOMER_POOL_INFORMATION ストアド・プロシージャに ObtainCustomerPoolInformation という名前を割り当てています。返却されるオブジェクトは Pool です。@StoredProcedureParameter アノテーションのリストを指定すると、このストアド・プロシージャに複数のパラメータや戻り値を割り当てることができます。

さらに複雑なシナリオでは、ストア・プロシージャに対して複数の結果のマッピングが必要になる場合もある。

るでしょう。その場合は、アノテーション内で`resultSetMappings`属性を指定し、`@SqlResultSetMapping`の名前のリストを渡します。

@NamedStoredProcedureQuery
で宣言したストアド・プロシ
ージャを呼び出すには、**リスト9**
に示すように、ストアド・プロシ
ージャに割り当てた名前を渡し
てエンティティ・マネージャの
createNamedStoredProcedure
Query()を呼び出します。返却さ
れるStoredProcedureQueryオブ
ジェクトにパラメータを渡すた
めには、setParameter()を呼び出
して位置と値のペアを指定し
ます。結果は、StoredProcedureQuery
getOutputParameterValue()メソ
ッドで取得できます。パラメータを手
動で登録したい場合や、動的に登
録したい場合は、registerStoredProcedureParameter(position, type, ParameterMode)メソッドを使用し
ます。

エンティティ・マネージャを介したストアド・プロシージャ・クエリー：ストアド・プロシージャは、エンティティ・マネージャの `createStoredProcedureQuery()` メソッドに、データベースに格納されているプロシージャ名を渡すことによってマッピングすることもできます。その後は、`setParameter()` メソッドを使用してリスト9と同様にパラメータの登録や戻り値の割当てを行うことができます。

確認のため、**リスト10**では、
先ほどの例と同じストアド・プロ

リスト7 リスト8 / リスト9

```


public static void obtainCustomerPoolInformation(
    long customerId,
    String[] poolInformation) throws SQLException {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet res = null;

    try {
        // "jdbc:default:connection"は
        // DriverManagerに既存のコネクションを使用することを伝えます
        conn = DriverManager
            .getConnection("jdbc:default:connection");
        // 問合せの作成
        String sql = "SELECT STYLE, SHAPE, LENGTH, " +
            "WIDTH, RADIUS, GALLONS " +
            "FROM POOL P, CUSTOMER CUST " +
            "WHERE P.ID = CUST.POOL_ID " +
            "AND CUST.CUSTOMER_ID = ?";
        stmt = conn.prepareStatement(sql);
        stmt.setLong(1, customerId);
        res = stmt.executeQuery();

        poolInformation[0] =
            (res.next()) ? res.getString(1) : "N/A";
        ...

    } finally {
        // クリーンアップの実行
        ...
    }
}

```

 すべてのリストのテキストをダウンロード

シージャを使用しています。この方法では、エンティティ・クラスに `@NamedStoredProcedureQuery` アノテーションを追加する必要は

ありません。単に、データベースに格納されているプロシージャ名をエンティティ・マネージャの `createStoredProcedureQuery()` メ

ソッドに渡してストアド・プロセスの呼出しを初期化するだけです。

SQL問合せ、更新、問合せ拡張

JPA 2.1でもっとも更新が多い部分は、SQLに関する部分かもしれません。いくつかの改善によって、基盤となるデータソースへのデータの問合せ、更新、削除に関して、今まで以上に多くのオプションが使いやすくなっています。

コンストラクタ結果マッピング：コンストラクタ結果マッピングは、`@SqlResultSetMapping`アノテーションを使用して、Javaオブジェクトのコンストラクタに問合せの結果をマッピングする機能です。この機能を使用して、エンティティに問合せの結果を設定することができます。この処理を実装するためには、`@SqlResultSetMapping`アノテーションで`name`を指定し、`classes`属性に、詳細を設定した`@ConstructorResult`を含めます。

@ConstructorResult アノテーションの目的は、対象のクラスを @ColumnResult アノテーションで指定した複数の列にマッピングすることです。列は、対象クラスのコンストラクタの引数と同じ順序になるように記述します。**リスト11a**と**11b**にコンストラクタ・マッピングを行うエンティティの構築方法を、**リスト12**にネイティブ問合せでオブジェクトにデータを設定する方法を示します。

動的名前付き問合せ:アプリケーションによっては、名前付き問合せを動的に作成する必要があります。その

ような場合でも、JPA 2.1が発表される前は、名前付き問合せを動的に作成することはできませんでした。JPA 2.1では、`EntityManagerFactory.addNamedQuery()`を使って動的な生成を行うことができるようになりました。

`addNamedQuery(String, Query)`
メソッドは、問合せに割り当てる名前の文字列と、動的に生成できるQueryオブジェクト自体を引数に取ります。動的な名前付き問合せを生成する例を、**リスト13**に示します。


JPA Query Languageの拡張

JPA Query Language (JPQL) にもいくつかの改善が行われており、さらに複雑な問合せが可能になっています。たとえば、問合せを細かく制御できるよう、外部結合 (**JOIN**) 条件に基づく表の問合せに **ON** キーワードが使用できるようになっています。外部結合を使用すると、標準結合で得られる結果を拡張できます。外部結合は、結合条件を満たすすべての行に加え、片方の表の条件を満たさない行を返します。

JPQLで外部結合を行う構文を**リスト14**に示します。これは、プールを所有しており、現在メンテナンス契約を行っている顧客の数を返します。この例では、主キー/外部キーのリレーションシップに基づいて2つのエンティティを結合しています。1対多関係があるので、各Poolには、Customerオブジェクトのコレクションが含まれています。この例では、ONによる結合条件の指定にcurrentMaintenanceフィールドを使用しています。

リスト10 / リスト11a / リスト11b / リスト12 / リスト13

```
StoredProcedureQuery spq =
    em.createStoredProcedureQuery(
        "OBTAIN_CUSTOMER_POOL_INFORMATION"
    );
spq.registerStoredProcedureParameter(
    1, Integer.class, ParameterMode.IN);
spq.setParameter(1, customerId);
return spq.getResultList();
```

 すべてのリストのテキストをダウンロード

次の例は、JPQL問合せの中からデータベース関数を呼び出すことができる**FUNCTION**キーワードです。データベース関数には、必要に応じ

てパラメータを渡すこともできます。
このJPQLの新機能を使用すると、簡潔で便利な構文で定義済みデータベース関数やユーザー定義データ

ノテーションでは、discriminatorカラムの名前と型を指定します。

この例では、`Pool`エンティティは `AboveGround`と`InGround`という2つのエンティティに継承されています。各サブクラスには、サブクラスの `discriminator`カラムとして使用する値を指定する `@DiscriminatorValue` アノテーションが含まれています。`AboveGround`エンティティは、次のように定義されています。

```
@Entity
@DiscriminatorValue("ABOVE")
public class AboveGround
    extends Pool {
    public AboveGround(){
        ...
    }
}
```

TREAT構文を使用するのは、AboveGroundオブジェクトとして扱われるすべてのPoolエンティティを返す問合せを作成したい場合です。**リスト17**に使用方法を示します。

Criteria APIの一括操作: Criteria APIも拡張され、一括更新と一括削除の機能が追加されました。Criteria APIを使用するためには、`EntityManager`から`CriteriaBuilder`オブジェクトを取得し、それを使用して、指定したエンティティ・クラスを扱うことができるオブジェクトを生成します。JPA 2.1では、一括更新操作を行う`CriteriaUpdate`オブジェクトと、一括削除操作を行う`CriteriaDelete`オブジェクトがAPIに追加されています。これらのオブジェクトも、`CriteriaBuilder`

リスト14 / リスト15 / リスト16 / リスト17

 すべてのリストのテキストをダウンロード

リスト18のコードには、`updateMaintenanceByCity()`という名前のメソッドが含まれています。このメソッドは、Criteria APIによる一括更新操作の実装例を示しています。リストでは、指定された都市のすべての`Customer`インスタンスに対して、現在のメンテナンス状態を設定して更新します。

するすべてのCustomerインスタンスを更新します。

削除の場合も、CriteriaUpdateオブジェクトの代わりにCriteriaDeleteオブジェクトを使用して、ほぼ同じ方法で実行できます。このCriteria APIの拡張によって、JPQLは使用せずに、完全にCriteria APIだけでアプリケーションを開発することもできるようになりました。

エンティティ・グラフ:以前は、エンティティでは、リレーションシップのロード手法をEAGERまたはLAZYのどちらかに指定する必要がありました。つまり、エンティティは、ロード手法を静的に制限されており、すべてのリレー



本記事では、JPA 2.1で導入されたすべての新機能を説明してはいません。エンティティ・リスナーにBeanをインジェクションする機能、`@PreDestroy`メソッドや`@PostConstruct`メソッドを実装する機能など、他にもいくつ

</article>

 [すべてのリストのテキストをダウンロード](#)

- [Java EE 7チュートリアル](#)
- [AcmePoolsのソースコード](#)
(GitHub)

このアプリケーションのmainメソッドを**リスト2**に示します。処理内容の概要は、次のとおりです。

- 接続されているUSB重量計の検出
- 重量計との接続をオープン
- 60個の計測値を取得するために、データ取得要求を送信
- 重量計をクローズ

エラーや例外が発生した場合でも、適切に重量計をクローズできるように `try finally` ブロックを使用しています。

リスト3は、最初に呼び出される **findScale** メソッドの実装です。このメソッドは、JavaX-USB APIを使用してDymo M10またはM25重量計を検出しています。メソッドの内部では **findDevice** メソッドを呼び出しています。このfindDeviceメソッドには、USBデバイス・ツリーを探索するコード (**リスト4**) が記述されています。

重量計からのデータを読み取るには、適切なインタフェースとエンドポイントに接続する必要があります。しかし、USB重量計のプロトコルはかなりシンプルであり、ここでは単に最初のインタフェースとエンドポイントを取得し、重量計から直接送信されてくるデータの待ち受けを開始するだけです。**リスト5**にこの処理を示します。

ここで注意点ですが、`UsbInterfacePolicy`を引数として`claim`メソッドを呼び出す必要があります。この呼び出しにより、カーネルがUSBインタフェースから強制的にデタッチされ、アプリケーションがUSBインタフェースを要求できるようになります。

syncSubmitの実装は簡単で、UsbPipeにある同名のメソッドを呼

び出すだけです。

```
private void syncSubmit()
    throws UsbException {
    pipe.syncSubmit(data);
}
```

ただし、実際の処理はコールバック時に行われます。コールバックを受けるためには、`UsbPipeListener`を実装し、2つのメソッドを準備する必要があります。最初のメソッドは`dataEventOccurred`です。これは、`syncSubmit`を実行した結果として呼び出されるもので、重量計から返されるデータを扱います。2番目のメソッドは`errorEventOccurred`で、重量計とのインタフェースに問題が発生した場合に呼び出されます。

出荷重量計から送られたデータは、6つの値を含む単純なバイト列です。プロトコルは正式に文書化されてはいませんが、オープンソース・コミュニティによってリバースエンジニアリングが行われています。各バイトの意味は次のとおりです。

- バイト0—未使用
- バイト1—特殊フラグ：空=2、重量超過=6、負=5（重量超過と負によって示される条件については後述）
- バイト2—計測単位：グラム=2、オンス=11
- バイト3—重量の倍率
- バイト4—基礎重量の下位バイト
- バイト5—基礎重量の上位バイト

リスト6はdataEventOccurredの実装です。ここでは、返されたバイト列を分解し、重量計の値を人間が読める形式でコマンドラインに表示して

リスト1

リスト2


リスト3

リスト4

リスト5

リスト6

```
javax.usb.services = org.usb4java.javax.Services
```

 すべてのリストのテキストをダウンロード

います。

その際、重量の大きさを得るためにビットシフト演算を行う必要があります。また、**リスト7**で示すように、バイト3で返された`scalingFactor`を使って倍率を考慮した重さを計算します。

エラーが発生した場合は、ログに記録して処理を続行するとよいでしょう(**リスト8**)。

最後のメソッドは、重量計のテストを終了するcloseアルゴリズムです。これは、次に重量計を使用するアプリケーションが重量計にアクセスできる

ように、単純にパイプとインタフェースをクローズしています。

```
public void close() throws
    UsbException {
    pipe.close();
    iface.release();
}
```

完成した`UsbScaleTest`クラスを実行すると、通常と同様にコンパイルが行われてJARファイルがデプロイされますが、それに続いて出力結果が表示されます。重量計が接続されて電

源も入っている場合は、重量計の最初の60回分の読取り値が出力されます。また、重量計が空、重量超過、負であるかどうかも出力されます。出力の例を**リスト9**に示します。

USBインタフェースがどのように動作するのか理解を深めるため、重量計がデータを返している間に重さの異なる物をいくつか置いてみましょう。負の値を表示したければ、物を重量計に置いてTARE（重量をゼロにする）ボタンを押し、それから置いた物を取り除きます。重量超過を試すには、重量計がサポートする重さよりも重い物（たとえば、Dymo M10重量計の場合は11キログラムの物）を乗せます。ただし、重量計のデリケートな内部コンポーネントが破損する可能性があるので、重すぎる物は乗せないように注意してください。

次は、ここまでの成果を応用して完璧な比率のコーヒーを入れます。そのためには、コーヒーの分量計算が必要になります。

完璧なコーヒーを計算する

本アプリケーションで利用する科学的知識の背景となっているものは、アメリカ・スペシャルティ・コーヒー協会（SCAA）のコーヒー抽出管理チャートです。これは、コーヒーを入れる際に最適な粉の量を計算するためのものです。このチャートは、Earnest Earl Lockhart博士のグループが1960年代に行った研究に基づいたものです。博士は、コーヒーを入れる際の諸条件を視覚的に表したことや、最適な濃さや抽出方法についての推奨ガイド

を作成したことで知られています。北米版のチャートを図1に示します。

チャートには、3つの系列が示されています。縦軸はコーヒーの濃さで、総溶解固形分(TDS)の割合で表されています。横軸は抽出比率で、割合で表されています。斜めの線は粉と湯の割合です。これは、リットルあたりのグラムで表されています。

ここで、今回のアプリケーションが対象とする「レギュラー」コーヒーを1リットルあたり55グラムと定義します。これは、抽出割合に対するSCAAの最適なバランスがもっとも得やすい値です。


なお、ヨーロッパ・スペシャルティ・コーヒー協会(SCAE)も同じ18%から22%の抽出比率を推奨していますが、最適な濃さは1.2%から1.45%の可溶性固形分としています。ここから、「リッチ」コーヒーを1リットルあたり58グラムと定義しましょう。さらに、ノルウェー・コーヒー協会では、最適な濃さを1.3%から1.5%の可溶性固形分とさらに高く設定しています。これを1リットルあたり62グラムの「ストロング」コーヒーと定義します。

リスト10に、**CoffeeCalculator**クラスを示します。このクラスは3つの基準をコード化したもので、完璧なコーヒーの分量を計算してくれます。

入れるコーヒーの量(カップの大きさ)を指定すると、CoffeeCalculatorクラスは使うコーヒーの粉の量を決定してくれます。なお、すべての値はグラムとします。ほとんどの重量計はグラムを使用しており、メートル法を使う方が水の重さを量に変換するのが

リスト7 リスト8 / リスト9 / リスト10

```
private double scaleWeight(int weight, int scalingFactor) {
    return weight * Math.pow(10, scalingFactor);
}
```

 すべてのリストのテキストをダウンロード

簡単であるためです。

下記は、このクラスの簡単な使用例として、300グラムの水に対する粉の使用量を計算しています。

```
grindWeight(REGULAR, 300)
```

このコードからは、17.4グラムという結果が返されます。人気のコーヒー店のレシピを試したことがある方なら、この数値が妥当であることが分かるでしょう。

非同期通信

ここまでのサンプルでは、重量計の読取りに同期アルゴリズムを使用しま

した。しかし、この方法はコーヒーのアルゴリズムに適したものではありません。なぜなら、USB重量計のデータはバッファリングされ、オンデマンドで読取りを行おうとしても、最初のうちは古いデータが返されるためです。重さを知りたいタイミングまで待機してから読取りを行おうとしても、バッファリングされている古いデータをすべて読み捨てない限り、現在の重量計の重さは取得できません。そこで、データを継続的に読み取ることによって、現在の重量計の値のスナップショットをリアルタイムに取得し、常にそれをメモリ内に保持しておくよ

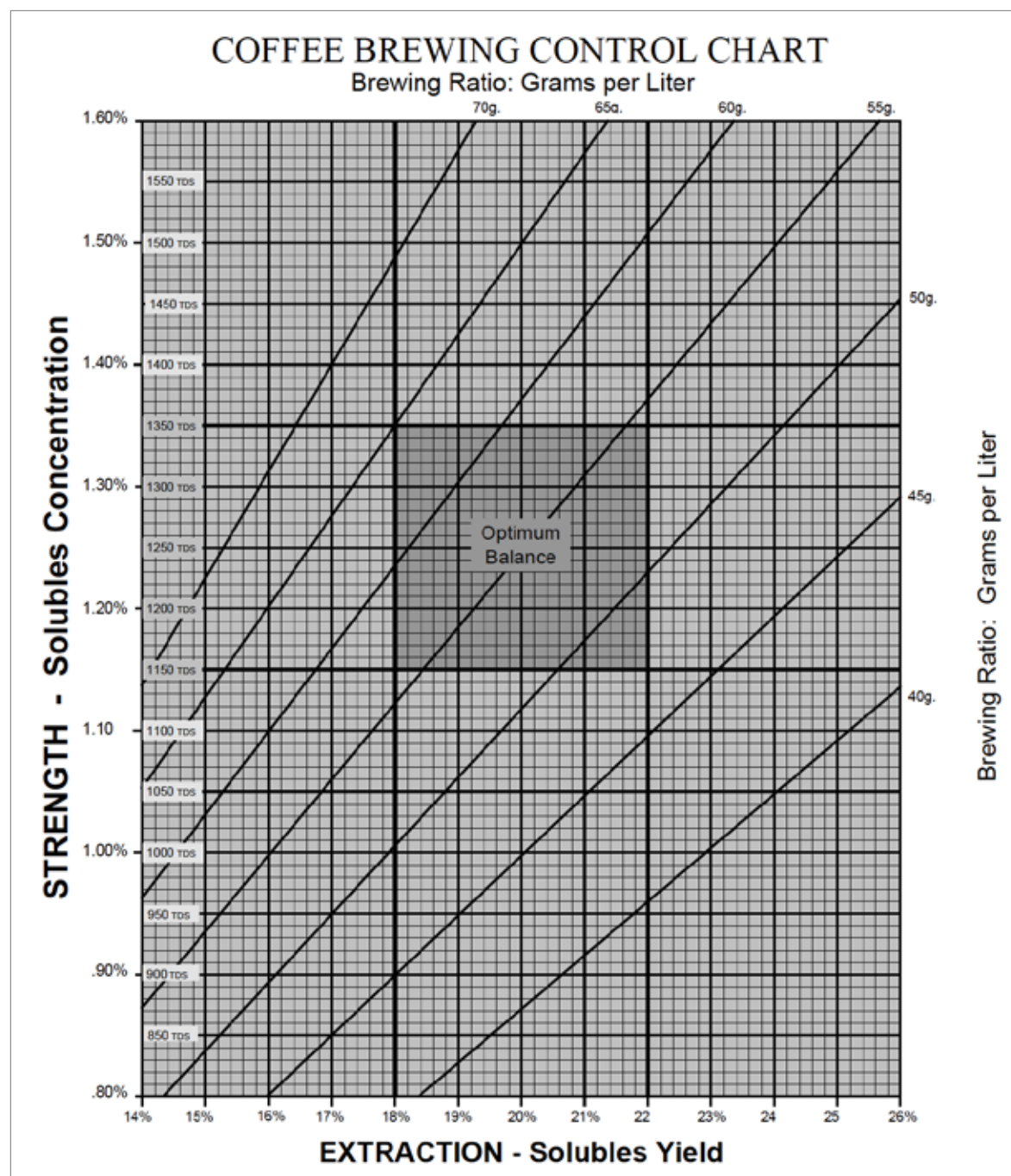


图1

うにします。重量計が特定の値を指すのを待つ場合は、次の値が届くまでスレッドを簡単にブロックできます。


リスト 11 のコードは、USBPipeListener コールバックを新たに実装したのですが、返された値

を処理し、データ要求を新たに送信し、`scalePhaser`上のブロック中スレッドに新しいデータを受信したことを通知しています。

Phaserは、Java SE 7で新しく導入されたスレッド制御クラスの1つです。1

リスト11

```
public void dataEventOccurred(UsbPipeDataEvent upde) {
    processData();
    if (closing) {
        busy = false;
    } else {
        try {
            pipe.asyncSubmit(data);
        } catch (UsbException ex) {
            errorEventOccurred(
                new UsbPipeErrorEvent(upde.getUsbPipe(), ex));
        }
    }
    scalePhaser.arrive();
}
```

 [すべてのリストのテキストをダウンロード](#)

つのスレッドからの到着通知でスレッド(または複数のスレッド)のブロックを解除できるため、前述した処理に最適です。本サンプルでは、次のようにパーティを1にしてPhaserを作成します。

入された`DoublePredicate`関数型インタフェースの使用法を示しています。たとえば、重量計が80グラム以上を読み取った場合など、任意の待機条件をラムダ式として渡すことができます。

```
private final Phaser scalePhaser =
    new Phaser(1);
```

```
scale.waitFor(w -> w >= 80);
```

1つしかパーティが登録されていないため、コールバックの度にPhaserからの通知が発生します。次に、これをブロックします。**リスト12**のように、現在のphaseの値を使ってawaitAdvanceを呼び出すだけの簡単なものです。

waitForメソッドは、Java 8リリースでラムダ式をサポートするために導

ここが、USB重量計の通信ロジック改善版の要点です。リファクタリングを行い、[UsbScale.java](#)という新しいクラスを作成します。このクラスは、[Scale](#)インタフェースとして統合された処理も実装しています。このインタフェースは、シリアル重量計の別の実装を作成する際にも使用できます。Scaleインタフェースの全メソッドを[リスト13](#)に示します。

ここで、コーヒーのレシピの信頼性と一貫性を改善するために、安定した測定値とプログラムによる風袋引き (TARE) という2つの重量計の概念について説明します。重量計は新しい重量を即座には認識しません。値が落ち着くまで激しく変動する場合もあるため、安定性という概念が重要になります。重量計の中には、データを通知する手段そのものに安定性の概念を取り入れているものもありますが、USB重量計はそうなってはいません。回避策として、**リスト14**のようにして、重量が更新された場合、直前のいくつかの結果と比較し、安定した状態になったかどうかを判断します。

こうすることで、目標の値に到達してから値が振れることはなくなり、値が安定します。その結果、レシピに沿ってコーヒーを入れる際に、信頼性の高い数値を得ることができるよう



图2

になります。この例では、`pastWeights`配列のサイズを調整することで、値を安定させるための重量比較の回数を制御できます。スピードと正確性のトレードオフになりますが、過去2回分の読取り値を現在の測定重量と比較するくらいがよいでしょう。

```
private final int[] pastWeights =
    new int[] {-1, -1};
```

重量計の値をゼロにリセットする操作も、レシピに従ってコーヒーを入れるうえで重要な要素です。この操作は、風袋引きと呼ばれます。重量計の中には、コマンドを送るとプログラムの風袋引きを行うものもありますが、今回のUSB重量計はそれには該当しません。そのため、ここでは、ユーザーに手動でTAREボタンを押してもらうようにメッセージを表示することだけです。重量計でプログラムによる風袋引きがサポートされていないことは、組込みの `UnsupportedOperationException` で伝えます。

```
@Override
public void tare() {
    throw new
    UnsupportedOperationException();
}
```

ScaleクラスとUsbScaleクラスのソース・コードは、[GitHubリポジトリ](#)から完全版をダウンロードできます。

コーヒーを入れるレシピ

リスト12 リスト13 / リスト14

```
public void waitFor(DoublePredicate condition) {
    while (!condition.test(getWeight())) {
        scalePhaser.awaitAdvance(scalePhaser.getPhase());
    }
}
```

 [すべてのリストのテキストをダウンロード](#)

以上で、USB経由で接続された重量計と非同期に通信できるようになりました。次のステップは、[CoffeeCalculator](#)クラスと[UsbScale](#)クラスを使用してコーヒーを入れるレシピを作成することです。コーヒーを入れる方法は何でもかまいませんが、ここでは、**図2**に示す[Aerobieエアロプレス・コーヒーメーカー](#)をお勧めします。これは、シンプルで安価なポータブル・コーヒーメーカーで、持ち運びにも便利です。

エアロプレスの使い方ですが、まずコーヒーの粉とお湯の一部を本体に入れます。本体で粉をかき混ぜると直接コーヒーが抽出されるので、60秒ほど待ちます。次に、プランジャーを差し込み、圧力を加えて空間を確保し、コーヒーがフィルタを通るようにします。これで、エスプレッソのよう

に濃縮されたコーヒーが抽出されます。お湯を加えて薄めれば、お望みの濃さのコーヒーのできあがりです。

エアロプレスで入れるコーヒーには、フレンチ・プレスで入れたコーヒーのように均一でありながら、フィルタで入れたコーヒーのように固形物が残らないという2つのメリットがあります。そのため、粉の粒の大きさや均一性はあまり気にする必要はありませんが、コーヒー豆が加熱されることがないように、高品質の円錐形バー・グラインダーを使用することをお勧めします。まだ電動のグラインダーをお持ちでなければ、**図3**に示す日本製の手動グラインダー Porlex Miniがおすすめです。持ち運びが簡単なエアロプレスとの相性も抜群です。一定の大きさで、しかも高品質にコーヒー豆を挽け



图3

るだけでなく、クランク・ハンドルを外せばエアロプレスの本体にぴったり収まる大きさになっています。

レシピの再利用を考慮し、まずは土台となる部分を作成しましょう。最初に必要になるのはRecipeインタフェースです。エアロプレスでのコーヒーの入れ方、そして将来作成するかもしれない別のレシピをどのように記述するかは、このインタフェースで決まります。Recipeインタフェースには、次の4つのメソッドがあります。

```
public interface Recipe {
    String name();
    String description();
    Ingredient[] ingredients();
    Step[] steps();
}
```

最初の2つのメソッドは、作成する

Recipeの名前と説明を返すものです。3つ目のメソッドは、ラッパー・クラスを使用して材料のリストを返します。最後のメソッドは、実際の一連の手順です。直接Javaコードで手順を記述することも可能ですが、このメソッドを使用すれば、手順をスキップしたり繰り返ししたりするなど、後でレシピのフローを改善できます。

IngredientクラスとStepクラスは簡単です。どちらもファクトリ・パターンに従っており、クラスのインスタンスを作成する静的メソッドを持っています。**リスト15**に示すように、材料はCoffeeCalculatorクラスを使用して重量で指定します。このコードは、AeroPressCoffeeレシピ用に材料を初期化しています。

このアルゴリズムは、一定の分量でコーヒーを入れるようになった

リスト15

```
public AeroPressCoffee(double strength) {
    beans = Ingredient.byWeight(CoffeeCalculator.grindWeight(
        strength, CUP_SIZE), "Coffee Beans");
    brewingWater = Ingredient.byWeight(beans.getWeight() /
        BREW_RATIO, "Water");
    extraWater = Ingredient.byWeight(CUP_SIZE -
        brewingWater.getWeight(), "Water");
}

@Override
public Ingredient[] ingredients() {
    return new Ingredient[]{beans, brewingWater, extraWater};
}
```

 [すべてのリストのテキストをダウンロード](#)

ています。分量はCUP_SIZE定数として設定されており、単位はグラム(ミリリットル)です。コンストラクタに濃さが渡されますが、その値は、CoffeeCalculatorクラスに定数として定義されており、レギュラー、リッチ、ストロングのいずれかのコーヒーに対応します。

エアロプレスでコーヒーを入れる場合、もう1つ **BREW_RATIO** という定数が出てきます。これは、コーヒーの粉と最初に混ぜ合わせるお湯の量を定義する定数で、抽出プロセスに影響します。コーヒーの苦みが強すぎる場合（お湯が多すぎるため、過抽出された）や酸味が強すぎる場合（お湯が足りず、十分抽出されていない）は、このパラメータを調整してコーヒーの粉に通すお湯の量を変更しま

す。2つの定数の推奨値を**リスト16**に示します。

最後に、もっとも重要な [AeroPressCoffee](#) レシピに取りかかります。このレシピは、**リスト17**に示すように、一杯のコーヒーをJavaで完璧に入れるプロセスを19段階の手順で構成しています。GitHubリポジトリからソース・コードの完全版をダウンロードし、実際に試してみることができます。


メイン・クラスは `JavaScale` です。このクラスは、`CommandLineRecipeRunner` を実行し、コーヒーのストロングを指定して `AeroPressCoffeeRecipe` を呼び出しています (**リスト18**)。 **リスト19** に、デフォルトの設定で `JavaScale` アプリケーションを実行した際の出力を示します。

- コーヒーの濃さを変えてみます (REGULARまたはRICH)。
- 入れるコーヒーの量を調整します (ただし、エアロプレスには抽出プロセスに使えるお湯の量に物理的な限界があります)。
- コーヒー豆の種類や好みに応じて BREW RATIO を微調整します。

もっとも重要な部分: レシピは、**一杯のコーヒー**をJavaで完璧に入れるプロセスを19段階の手順で構成しています。

うまいければ、
コーヒーを飲む
のと同じくらい、
コーヒーを入
れるプログラ
ムの作成も楽し
めたかもしれま
せん。他にも組
込みJavaで実
現できる様々
な事例がある
ので、まもな
く発刊される
McGraw-Hill
の『[Raspberry](#)
Pi with Java:

```
private static final double CUP_SIZE = 300; // grams
private static final double BREW_RATIO = .2; // beans/water
```

 すべてのリストのテキストをダウンロード

- [GitHubリポジトリ](#)
- [usb4java \(USBデバイスにアクセスするためのJavaライブラリ\)](#)
- JSR 80 (Java USB API)



ANTONIO GONCALVES

パート1

Contexts and Dependency Injection: 新しいJava EEツールボックス

依存性注入で強い型付けを利用する

Antonio

Goncalves:

Java/Java EEを専門とする上級開発者。Java EE 5やJava EE 6、最近ではJava EE 7に関する書籍を執筆。Paris JUGおよびDevoxx Franceを設立したJava Champion。各種JSRの独立系JCPメンバーを務める他、PluralsightではJava EEのトレーニングを開講。

依存性注入に関する解決策の歴史

は古く、Javaの登場以前から存在しています。現在、Javaエコシステムには、Spring、JBoss Seam、Guiceなどの商用製品やオープンソース製品が展開されています。これらのフレームワークは成功を遂げたものの、標準化されることはありませんでした。Java EE 6にバンドルされているContexts and Dependency Injection (CDI) 1.0 (JSR 299) は強い型付けの方式を採用し、JBoss Seamの設計原則を多く取り入れました。それ以降、CDIはバージョン1.1 (JSR 346) にアップデートされた上でJava EE 7に搭載され、さらにメンテナンス・リリースが提供されています。CDIには複数の実装 (Weld、Apache OpenWebBeans、Caucho Resin CanDI) があり、Apache DeltaSpikeプロジェクトによって機能強化されています。このプロジェクトは、CDIの拡張機能を提供しています。

型保証されたインジェクション、オルタナティブ、インターセプト、デコレート、イベント管理、スコープ、コンテキストAPIの機能を含むCDIは、Java EEツールボックスとして利用されるようになりました。CDIは、ビジネス・アプリケーションを容易に開発するため、あるいは社内用の技術フレームワークを拡張するために使用できます。

本記事では型保証されたインジェクションに着目し、依存性注入における強い型付けの意味について確認します。

プログラムによるインジェクション

アプリケーションは、ビジネス・ロジック、他のシステムとの相互作用、ユーザー・インタフェース、外部APIなどから構成されます。実際、オブジェクト指向プログラミングではこれらの機能がクラス単位でまとめられ、各クラスは他のクラスに依存します。そのようなクラスの関係は強い結合(例:実装に依

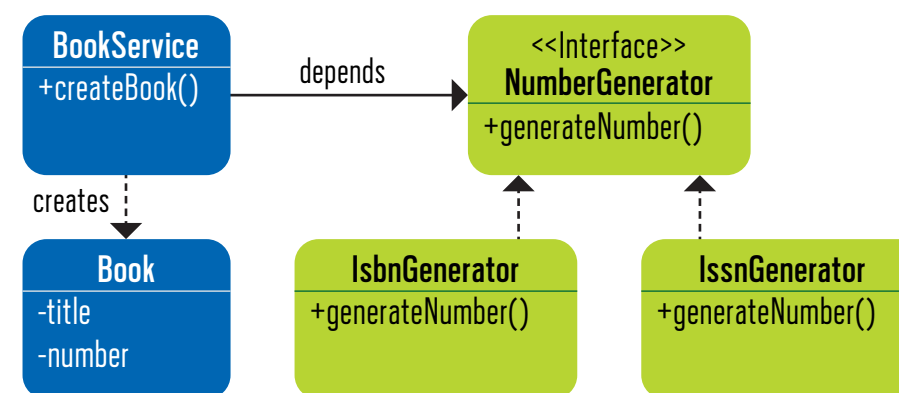


图1

存)の場合と、疎結合(例:インタフェースに依存)の場合があります。具体的な例を確認しましょう。

図1に示すBookServiceクラスの役割は、Bookオブジェクトを作成することです。1冊の本には、タイトルと番号が1つずつ含まれます。この番号は、ISBN番号 (IsbnGeneratorクラスによって生成される13桁の番号) またはISSN番号 (IssnGeneratorクラスによって生成される8桁の番号) のいずれかです。

Book Service は

NumberGeneratorインタフェースに依存します。このインタフェースには、generateNumberというメソッドが1つ含まれます。BookServiceは何らかの条件や環境に応じて、IsbnGenerator実装またはIssnGenerator実装のいずれかを選択できます。コーディングの選択方法としてまず考えられるのは、実装をコンストラクタに渡して(**リスト1**)、実際に使用する実装については外部のクラス(Mainクラス)に選択させると

ATMs, Smartcards, POS Terminals, Blu-ray Players,
Set Top Boxes, Multifunction Printers, PCs, Servers,
Routers, Switches, Parking Meters, Smart Meters,
Lottery Systems, Airplane Systems, IoT Gateways,
Programmable Logic Controllers, Optical Sensors,
Wireless M2M Modules, Access Control Systems,
Medical Devices, Building Controls, Automobiles...



#1 Development Platform

ORACLE®

```
@Alternative
@ThirteenDigits
public class MockGenerator implements NumberGenerator {

    public String generateNumber() {
        return "mock-" + Math.abs(new Random().nextInt());
    }
}
```



まとめ

CDIは、デフォルト・インジェクションなどの単純なインジェクションのケースに対応しますが、修飾子を使用して複雑なケースを解決することもできます。修飾子は、あいまいな依存性を解決するうえで、型保証された方式として使用します。修飾子はアノテーションです。したがって、メンバーの追加や別の修飾子との併用によって、一意に定義されたインジェクション・ポイントを構成できます。候補は、beans.xmlデプロイメント・ディスクリプタに記述して、デプロイ時にインジェクション・ポイントを調整するために使用できます。本記事で確認したとおり、CDIはインジェクションの面では機能豊富であり、型保証され、使いやすいAPIを提供しています。

本記事では、CDIに関する4つのパートから成るシリーズの第1回をお届けしました。第2回では、プロデューサを使用してサード・パーティ・フレームワークを統合する方法を説明します。第3回では、インターセプタ、

デコレータ、イベントを使用した疎結合の方法に着目します。最終回では、Java EEでのCDIの統合について説明します。

- [CDI仕様](#)
- [Weldリファレンス実装](#)
- 『Beginning Java EE 7』(書籍)



MICHAEL KÖLLING

パート2

Microsoft Kinectを使用した 簡単なピンポン・ゲームの作成

Greenfootを利用してより楽しくJavaからKinectを操作する

Michael Kölling: イギリスのカンタベリーにあるケント大学コンピューティング学部教授。研究分野はオブジェクト指向システム、ソフトウェア・ツール、プログラミング言語、コンピューティング教育、HCI。Java 教本2冊を出版。BlueJおよびGreenfootの開発リーダー。

写真提供：
ケント大学

Greenfootは、開発初心者を対象としたJavaベースのグラフィカル・プログラミング環境です。Java Magazineの2014年9/10月号では、GreenfootおよびMicrosoft Kinectを使用してプログラミングの初歩を学習しました。センサー・ボードとしてMicrosoft Kinectを使えば、カメラの前で人の身体が動く様子を追跡できます。GreenfootやKinectについてよく知らない方は、まず第1回の記事を読み、ハードウェア/システムのセットアップ方法、GreenfootからKinectへのアクセスに関する基本事項、および空中で手を動かして画面上にペイントするペイント・プログラムに関し必須となるプログラミング・タスク等、以上の詳細を確認してください。

本記事では、キーボードで操作する昔ながらのPong風ピンポン・ゲームを題材として、右手と左手に1つずつ、計2つのラケットを持って、ジェスチャに

よってゲームを操作できるようにします(図1)。

昔ながらのPong

まず、本記事の内容に沿ってプログラミングできるように、ベース・プロジェクトの[kinect-pong.zip](#)をダウンロードしてください。このGreenfootシナリオでは、初歩的なPong風ピンポン・ゲームを実装します。プレイヤーは画面の上部と下部のラケットで、ボールを上下に打ち合います。下部のラケットはプレイヤーがキーボードのカーソル・キーで操作し、上部のラケットはコンピュータにより制御されます。

Greenfootをインストールし、このシ

ナリオを開いてプレイしてください。既存のクラスのコードをひととおり確認してください。簡単に理解できると思います。

次に、Kinectをセットアップし、Kinectサーバーを起動します([前回の記事を参照](#))。これで、今回のシナリオのKinectプログ

ラミングを開始する準備が整いました。

ユーザーの検出

最初にすべきことは、ユーザーがKinectのカメラの前に現れたときに、それを検出することです。ゲームはユーザーがいな

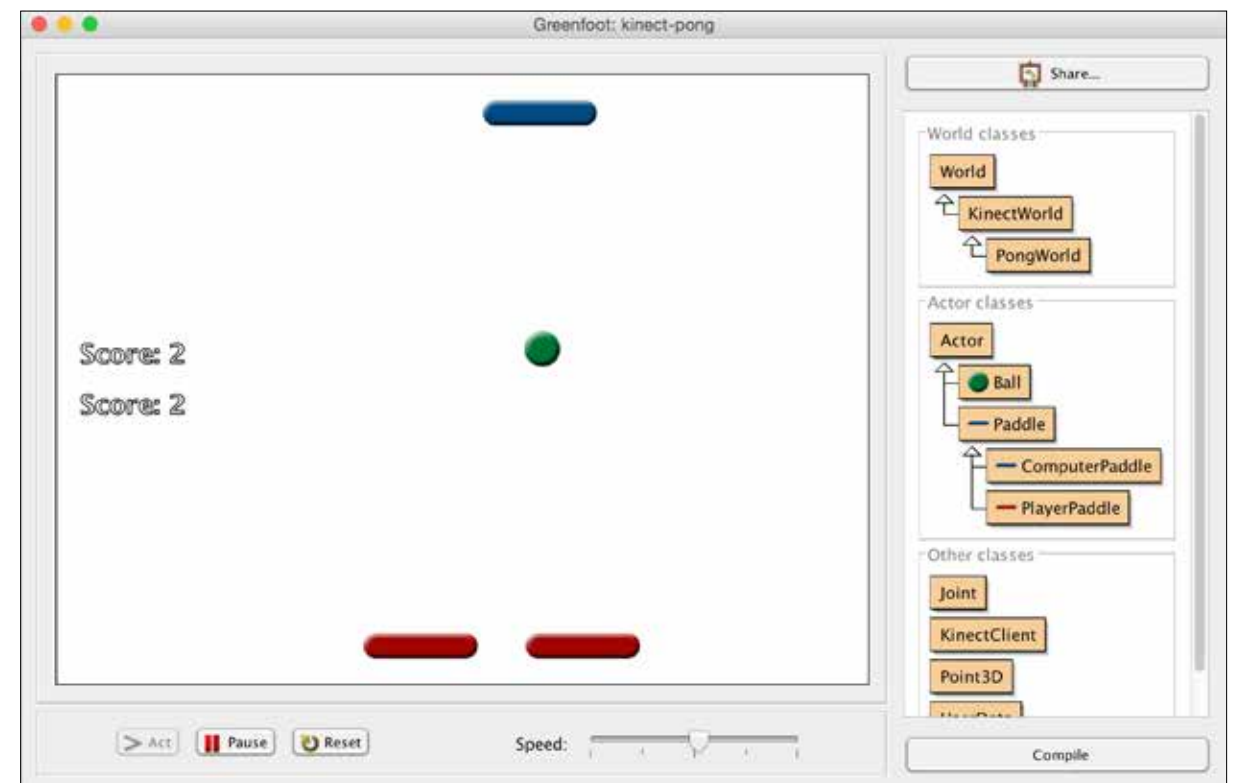


图1

い間はアイドル状態(待機中)となり、ユーザーがKinectの視野に入った時点でスタートします。このプロジェクトでは、この検出処理をPongWorldクラスで実装します。

`getTrackedUsers`メソッドで、追跡中の全ユーザーの配列を取得できます。

```
UserData[] users =
    getTrackedUsers();
```

さらに、この配列の長さが0であるかをチェックして、ユーザーが存在するかを確認できます。

```
if (users.length > 0) {
  ...
}
```

まずは、PlayerPaddleとBallを作成するコードを、PongWorldのコンストラクタから、startGameという新しいプライベート・メソッドに移動します。次に、ユーザーがゲームをスタートしたときに、actメソッドからこのプライベート・メソッドを呼び出すようにします。

```
if (users.length > 0) {
  startGame();
}
```

ただし、このコードには問題があります。`act`メソッドはシナリオの実行中に繰り返し呼び出されるため、このままでは多数のボールやラケットが作成されてしまいます。プレイヤーが最初に視野に入ったときの1回だけボールやラケットの作成を行うようにする必要があります。この目的で、クラ

スにすでに定義されているidleフラグ
を使用します。

```
if (idle) {
    if (users.length > 0) {
        startGame();
        idle = false;
    }
}
```

これまでに作成したコードは
リスト1のようになります。

ラケットの操作

現時点では、このゲームはカメラの前
にいるプレイヤーを検出しますが、プ
レイヤーはラケットを操作できませ
ん。その点をこれから改修します。

ラケットでプレイヤーの情報を利用するためには、追跡中のユーザーのデータを `PlayerPaddle` オブジェクトに渡す必要があります。このゲームは1人のプレイヤーを対象とするため、(複数のユーザーがいる場合は2人目以降を無視して) 追跡中ユーザーの配列のうち先頭だけを `startGame` メソッドに渡します。

```
startGame(user[0]);
```

さらに、startGameメソッドの定義に適切なパラメータを追加し、そのパラメータをPlayerPaddleコンストラクタに渡します。この新しいバージョンのstartGameメソッドはリスト2のようになります。

次に、PlayerPaddleクラスを変更します。PlayerPaddleのコンストラクタ定義に、UserDataオブジェクトを受け取るためのパラメータを追加し、受

リスト1


```
public class PongWorld extends KinectWorld
{
    private boolean idle;

    /* Create the game world with the game objects in it.*/
    public PongWorld()
    {
        super ();
        addObject(new ComputerPaddle(), 200, 30);
        idle = true;
    }

    /* Nothing to do. */
    public void act()
    {
        super.act();
        UserData[] users = getTrackedUsers();


        if (idle) {
            // check whether a user has arrived
            if (users.length > 0) {
                // discovered a new user
                startGame();
                idle = false;
            }
        }
    }

    private void startGame()
    {
        addObject(new PlayerPaddle(), 200, 450);
        addObject(new Ball(), getWidth()/2, getHeight()/2);
    }
}
```

 [すべてのリストのテキストをダウンロード](#)

本記事の例は、『[Introduction to Programming with Greenfoot](#)』（Pearson、2015年）で使用されているものです。この書籍には、GreenfootによるJavaプログラミングについて詳細な説明があります。リンク先で、他のGreenfoot Kinectプロジェクトも取得できます。</article>

- [Greenfootダウンロード・ページ](#)
- [GreenfootとKinectのセットアップ方法](#)
- [Greenfootの書籍](#)

 [すべてのリストのテキストをダウンロード](#)

BRUNO SOUZA,
EDSON YANAGA

Bruno Souza: Java 開発者。Summa Technologiesではオープンソース・エバンジェリストとして、ToolsCloudではクラウドの専門家として従事。SouJavaとWorldwide Java User Groups Communityの設立者。

Edson Yanaga: Produotec Informáticaの技術リーダー、Ínsula Tecnologiaのプリンシパル・コンサルタント、オープンソースのユーザー、提唱者、開発者。

BRUNO SOUZAの写真:BOB
ADLER/GETTY IMAGES

パート2

開発者として1段階上の キャリアを目指すためのアイデア

コードを学習し、カンファレンスに参加して人脈を構築する

このシリーズの最初の記事(2015年1/2月号、25ページ)では、開発者としてのキャリア向上につながる注目点として、3つの大きな領域を紹介しました。一番重要なのはコードです。これは、開発者としての基本スキルです。しかし、その他の2つであるコミュニティと発想の転換(いつもとは違う考え方)にも背を向けるわけにはゆきません。この2つは、キャリアに立ちふさがる壁と一緒に打ち破ってくれます。

この3つの領域に毎週いくらかの時間を費やすことで、優れた開発者への道を進んでいくことができるでしょう。プロフェッショナルとしての完成度を上げ、新しいチャンスを手にすることにもつながるはずです。キャリアをステップアップさせ、新たな道を見つける上ですべてのことが役立つのです。また、この記事で提案しているのは難しいことでも時間がかかることでもありません。忙しい日々を送っていても、時間を見つけて実行できるはずです。

コード:オープンソース・ソフトウェア

コーディングに秀でるためには、たくさんのコードを書く必要があることはすでに見てきました。しかし、コードを書くにしても、書いているのがよいコードなのか悪いコードなのかをどう判断すればよいのでしょうか。現在のところ、現実的で議論の余地がないコードの品質測定方法はありません。ほとんどの場合、コードの品質は定量的というよりも定性的な問題です。あるコードが確実によいコードだと断言することはできませんが、他に比べてよいコードか悪いコードかは認識することができます。優れた開発者になるための正しい道を実践に進みたいなら、最善の方法は、他の開発者のコードを見て自分の成長度合いを推し量ることです。

ソフトウェアの世界では、ソフトウェアのソース・コードを見ることで、他人のコーディング方法を知ることができます。その結果、そのコードを分析し、実験し、場合によっては改善する

ことまでできます。しかし、そのためには、優れた開発者のソース・コードにアクセスする必要があります。

オープンソースのすばらしさはここにあります。コードを自由に入手でき、それを分析して自分のコードと比較することができるよう。 [GitHub](#)や[Bitbucket](#)を見れば、小さなものからエンタープライズ・レベルのものまで、莫大な数のプロジェクトが見つかります。そこから、新しい技術や、異なる視点を学ぶことができます。優れたコードを見て感動することも、経験の乏しい開発者のコードを見て反面教師とすることもできます。

コードをテストする方法を探したいなら、オープンソース・プロジェクトを調べて、他の開発者のテスト方法を調べてみましょう。システムでしかるべきログ出力の方法が分からない場合は、現在使用中のオープンソース・プロジェクトの中からログ情報の充実したものを選んで、同じ手法を修得しま

しょう。

オープンソースに貢献すれば、日々の仕事では使っていない(あるいは、使うことができない)スキルを修得することもできます。仕事として多くの人が行うのは生成、読取り、更新、削除(CRUD)機能を持った従来型エンタープライズ・システムの作成に限られるでしょう。しかし、余暇の時間を使えば、学習や貢献の対象として興味をそそるオープンソース・プロジェクトを自由に選ぶことができます。状況によっては、関連するオープンソースのツールやライブラリを仕事に利用し、バグの修正を行ったり、新機能を実装したりすることもできます。これは刺激的であるだけでなく、既存の考え方にとらわれない発想を行うための訓練にもなります。ドキュメントを追加したり、バグ・トラッカーで問題を提起したりするなど、小さいことから始めることができます。

やがては、バグ修正や新機能に関するプル・リクエストを送信できるようになるでしょう。そ

ORACLE®

現在SourcePointでCTOを務めているGeir Magnusson Jr.は、Apache Geronimo J2EEプロジェクトとApache Harmony J2SEプロジェクトの共同創設者でした。さらに、Apache Software Foundationにも、理事会のメンバーとして貢献しています。現在はJava Community Process (JCP) のExecutive Committeeに個人として参加しており、以前はApache Software Foundationを代表してJCP Executive Committeeに参加していました。

Java Magazine: Javaは20周年を迎えています。これまでのJavaプラットフォームの進化の中で、一番驚いたことは何でしょうか。

Magnusson: Javaが今でもここまでの価値を保っていることですね。開発者の注目を集めたテクノロジーで、Javaと同じように生き延び、発展してきたものはそうは思いません。

もちろん、ゲームの世界にはCやC++のように長い間存在し続けているプログラミング言語もあります。しかし、個人的な意見ですが、CやC++は新しいコン

SourcePointで同僚 の開発者と話し合う Magnusson

JCP Executiveシリーズ
JCPの内部でJavaを支える

SourcePointの**Geir Magnusson Jr.**が、Javaでシステムを構築するメリットと、JCPがどのようにJavaプラットフォームの進化に貢献してきたかを語ります。**STEVE MELOAN**

写真:MATT MILLS MCKNIGHT/GETTY IMAGES



「初めて見る人にも分かりやすいコードを書けるということは、極めて重要です」と Magnusson は語っています。

大きなヒープが必要でも、何の心配もいりません。ガベージ・コレクションに対する緻密な制御が行われているからです。メモリ管理について心配することはなくなりました。

さらに、Just-In-Time (JIT) コンパイルがまるで奇跡のように見えることもなくなりました。JVMは、バイトコードを非常に高速なマシン・コードに変換し、しかも完璧に最適化してくれますが、今ではそれが当たり前に感じられます。先ほどもお話したように、このすばらしく洗練されたソフトウェア・テクノロジーを、さまざまな言語がこぞって採用しています。

そんな中、JVMの改善は常に続いています。ガベージ・コレクション式の別の言語をいくつか使ったことがあります。JVMの方がはるかに優れています。

Java Magazine:Javaについて、払拭すべき誤解はありますか。

Magnusson: Javaは遅い言語だというのはよくある誤解です。これはまったくの間違いです。Javaは、高頻度で取引を行うコミュニティや、優れたパフォーマンスと非常に短い待機時間が要求される分野でも活用されています。

また、Javaの言語構造は、必要以上に冗長だと見なされることもあるようです。しかし、プログラミング言語は人間が読むためのもの

で、コンピュータ向けのものではありません。初めて見る人にも分かりやすいコードを書けるということは、保守性や明快さの面で極めて重要です。

それから、ときにJavaでのソフトウェア開発はペースが遅いという不満を耳にすることもあります。しかし、私の経験では、コードの記述、コンパイル、実行、テストといった一連の手続きは、とても高速に繰り返すことができます。IntelliJ、Eclipse、NetBeansのような最近のJava IDEを使えば、ほとんどのハウスキーピングを行ってくれるので、作業全体を効率化することができます。

最近はやりのビッグ・データ分野に注目すれば、ほとん

どのテクノロジーがJavaで書かれていることが分かります。オープンソース・ソフトウェア・フレームワークである[Hadoop](#)やその補完技術、オープンソース分散データベース・システムである[Cassandra](#)、さらに問合せおよび処理用のライブラリである[Cascalog](#)など、枚挙に暇がありません。Javaの環境は、制御し、調整し、理解することができます。すばらしいツールも存在し、プロファイリングにも優れています。数え切れない分野でアプリケーションが進化し続けていますが、それを支えているのがJavaなのです。

Java Magazine:あなたは以前、Apache Software Foundationを代表してJCP Executive Committeeに参加していましたが、今は個人として参加しています。この役割の違いや経験について、お話ししていただけませんか。

Magnusson: Apache Software Foundationを代表していたときは、私の個人的な関心事は当然多岐に渡っていました。しかし結局は、Apacheの問題が第一で最優先だったのです。

現在はJCPの個人会員なので、毎日Javaを使って革新的なソフトウェアを開発する人々の代表でありたいと思っています。内部利用または商用製品をオープンソースで提供する人々の代表です。

とりわけ関心があるのは、Javaエコシステムの安全性とオープンソースとの親和性を確かなものにする事です。私の意見では、現在のイノベ-

オープンソース

「私の意見では、現在のイノベーションの大部分は、オープンソース・コミュニティやオープンソース・プロジェクトに支えられています」

ス・プロジェクトのソフトウェアを使用すれば、標準Javaコンポーネントが一貫性をもって動作するという確信が得られます。Javaがここまで魅力的である理由の1つに、そうした安心感が挙げられます。開発者として、あらゆる要素がスムーズに協調し合うと自信を持って言えるからです。

イノベーションに関しては、パフォーマンス、システム統合、モニタリングなど、さまざまな側面に注力したさまざまなJVM実装があります。このフィールドには、色々なプレイヤーがいるのです。しかし、そういったJVMも標準Java環境であり、予測可能で信頼できるパフォーマンスを保証しています。

とはいえ、すべてに規格が必要なわけではありません。Hadoopソフトウェア・ライブラリは、市場で業界標準という地位を確立しています。これを正式な標準にしたところで、何の意味もありません。

ほとんどの点で、JCPはこの分野でよい仕事をしてきたと言えるでしょう。Javaがこの20年にわたって価値を維持してきた理由の1つはそこにあると思います。

正式な標準プロセスは、イノベーションの速度を落とすこともあるでしょう。しかし、ときにそれはよいことでもあります。ただ闇雲に最新の方法をとるのではなく、テクノロジーをしかるべき方向へと進化させることができるのですから。

Java Magazine: JCPは、進化するコミュニティにどのような形で力になれるのでしょうか。

Magnusson: 現在もっとも差し迫った問題は、JCPに参加することへの魅力を今以上に感じてもらえるようにすることと、Javaエコシステムの中のさらに多くの開発者に参加してもらうことです。もちろん、テクノ

ロジャー・ベンダーは引き続き代表者を派遣する必要があるでしょう。また、発言権や役割を持っている個人もいます。しかし、参加者の幅を広げる必要があるのです。JCPと開発者が日常的に連携できるよう新たな工夫を施す必要があります。

ほとんどのソフトウェアは、ソフトウェア・ビジネスを行っていない企業の開発者によって書かれています。建設会社、銀行、小売店などです。Javaが、そういったあらゆる企業や関連アプリケーションの分野に非常に適した存在であり続けることを確かなものにしなければなりません。

Java Magazine:あなたの新規事業と、その中のJavaテクノロジーの位置付けについて教えてください。

Magnusson:SourcePointという会社です。この会社は、Webやモバイルでコンテンツを発信する人向けの技術開発に焦点を絞っています。

近いうちに、発信者が広告ブロッカーを検知して、選択した方法で対応するツールを開発する予定です。たとえば、訪問者に対して、表示しようとしているコンテンツは広告によって運営されているため、広告ブロッカーをオフにするよう促すことができます。長期的には、発信者が訪問者と深いコミュニケーションをとれる技術を開発したいと考えています。

Javaに関して言えば、鍵となる役割を果たしています。私たちのフロントエンドのWebページはJavaScriptとRuby on Railsで

JCP
「もっとも差し迫った問題は、
開発者に JCP に参加すること
への魅力を今以上に感じても
らえるようにすることです」

実装していますが、ほとんどすべてのバックエンド・インフラはJavaで書かれています。インターネットからのデータ収集を行うのはJavaサーブレットで、あらゆるデータをデータ・パイプラインに送り込みます。

データ・パイプラインもJavaで書かれ、JVM上で実行されています。その後、Javaで書かれたカスタム・ソフトウェアでデータを処理し、CとJavaで書かれたデータベースに結果を格納しています。

Javaは長い間、私の開発プロセスにとって欠かせないツールであり続けています。JCPは、Javaがこれからも進化し続け、繁栄し続けることを保証します。

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

LEARN MORE

- ・JCPの個人会員になる



Christian Nicholson (左)、George Takeo (右) とともに、Skymind の主要製品についてミーティングを行う共同創業者の Adam Gibson (中央)

ディープ・ラーニング

起業まもないSkymindは、ディープ・ラーニング・テクノロジーを研究所から持ち出し、JavaとHadoopで構築したオープンソース・フレームワークと合わせて企業に持ち込もうとしています。

写真:BOB ADLER/GETTY IMAGES

ディープ・ラーニングとは機械学習の形態の1つで、あらゆる形状、サイズのインテリジェント・デバイスを使って人間の脳の学習機能をエミュレートしようという試みです。

このような機能は、自動運転車への応用など、ビッグ・データやモノのインターネットに莫大な見返りをもたらす可能性が

DL4Jフレームワーク

Deep Learning for Java (DL4J) はオープンソースのフレームワークで、エンタープライズ・コンピューティング環境でJavaを使ってディープ・ラーニング・アルゴリズムを開発できます。このフレームワークには、さまざまな人工ニューラル・ネットワーク (ANN) が含まれており、開発者はディープ・ラーニング・アプリケーションに解釈させたいデータの種類のに応じたANNをJava仮想マシン (JVM) に導入できるようになっています。ANNは統計的学習アルゴリズムとして、動物中枢神経系、とりわけ人間の脳のような生体神経回路網に近い機能を持っています。

一般的なANNには、いくつかのタイプがあります。畳込みネットワークは画像の学習によく使用され、再帰的ニューラル・テンソル・ネットワークは自然言語処理、テキスト・マイニング、文の解析に導入できます。

単純多層制約付きボルツマン・マシンである深層信念ネットワークは、複数のデータ型を扱います。ディープ・オートエンコーダ・ネットワークは、データ圧縮や、複雑な入力の次元を小さくする際に便利です。多層ノイズ除去オートエンコーダは、新しいデータを一般化するためにノイズ除去技術を活用しています。回帰ネットワークは、一般的に音声や時系列を学習するために使用します。

DL4Jフレームワークには、N-Dimensional Arrays for Java (ND4J) も含まれています。これは、NumPyおよびMATLABタイプのデータ分析ツールをJVMに導入する科学計算ライブラリです。[NumPy](#)はPythonプログラミング言語のエクステンションで、多次元配列と高水準数学関数の大規模ライブラリをサポートしています。[MathWorks](#)のMATLABは、マルチパラダイム数値計算環境であり、第4世代のプログラミング言語です。

DL4Jは、JVM上で動作する言語、Scalaもサポートしています。

の機械学習は、ディープ・ストラクチャル・ラーニングとも呼ばれており、人工ニューラル・ネットワーク(ANN)、パターン認識、統計、数理最適化や、その他の人工知能に関するテクノロジーを組み合わせたものです。

ディープ・ラーニングでは、ANN上で実行されるディープ・ラーニング・アルゴリズムがデータを適切に分類し、それに意味を与えます。その際に使用できる関連情報は、シグナルと呼ばれます。意味を与えらることは、写真やビデオから顔の特徴を認識したり、録音された音声から話し言葉を認識したり、文書やWebサイトに書かれた文字を認識したりすることです。その他の情報は、すべてノイズです。

「ディープ・ラーニング・アルゴリズムは

歷史

世界初のディープ・ラーニング
人工ニューラル・ネットワーク
は、1980 年に福島邦彦氏が
作成したネオコグニトロンだと
言われています。

データから自動的にシグナルを抽出し、そのデータを使って事前プログラミングを必要としない方法で問題を解決します。アルゴリズムがそのデータから多くのシグナルをキャッチできればできるほど、推論もしやすくなります。この点はビッグ・データ分析と同じです。これは、人間にとっては簡単



ドメイン名

.io トップレベル・
ドメインは、入出力
(I/O) を連想させ
ることから、とりわ
け最近のシリコン・
バレーの新会社の
間で人気です。

でも、コンピュータにとってはとても難しいことです。ディープ・ラーニング・アルゴリズムは、コンピュータが自動的に、人間とほぼ同じように情報を解析できるよう教え込むものです」とGibsonは述べています。

現在のディープ・ラーニング・アルゴリズムやANNは、PythonなどのJava以外のプログラミング言語で書かれていることがほとんどです。しかし、Gibsonにとっては、DL4JにJavaを使うのは自然な選択でした。現在、Javaは企業で重要な位置を占めているからです。彼はこう言います。「DL4JをJavaで開発することで、私たちは企業向け分野で有利なポジションを確保できます。企業と同じセキュリティ・メカニズムを使い、企業のプログラマーがすでに理解している環境で運用できるのです」

DL4Jは、[Hadoop](#)および[Spark](#)と統合されています。どちらも企業のビッグ・データ向け実稼働環境の業界標準であり、Apache Software Foundationから提供されています。SkyminDは、DL4Jにこの2つのソフトウェアのライセンス2.0を使用しています。HadoopはJavaで記述されており、SparkはScala (JVM上で動作する言語)で記述されています。「HadoopとSparkはJavaとともに企業のオープンソース・テクノロジーを牽引しています。この傾向は今後もしばらく続くでしょう」とGibsonは述べています。

ビッグ・データとその向こうにあるもの

DL4Jはさまざまなタイプのニューラル・ネットワークをサポートしているため、こ

のフレームワークをプラットフォームに用いてアプリケーションを開発すれば、ほぼあらゆるデータ型を解析できます(サイドバーの「DL4Jフレームワーク」を参照)。先行する他のオープンソース企業と同様、Skymindもトレーニング、実装、カスタマイズ、継続サポートといった有料サービスを提供しています。Skymindの重点領域は、テキスト分析や画像などです。

「テキスト分析は一般的なオフィスの課題で、私たちが多く相談を受けるのもこの分野です」とGibsonは言います。「カスタマー・サービスのログを保存する企業は多いですが、それをどうしたらよいのか、そのデータからどのようにシグナルを抽出すればよいのかを企業は知りません。たとえば、顧客がいつ不満を感じているかを判



Gibson、Nicholson、Takeo は、製品の機能やスケジュールを相談するため、頻繁にミーティングを行っています。



ERWAN PINAULT,
THALÍA CRUZ

Erwan Pinault は、Aldebaran で Java NAOqi SDK の開発に携わる Java エンジニアです。

Thalía Cruz は、Aldebaran でロボット・アプリケーションの開発者として働くソフトウェア・エンジニアです。Aldebaran に入る前は、フランスの新興企業 Onprint の Android 開発者でした。Cruz は、フランスの EPITA でコンピュータ・サイエンスの修士号を取得しています。

人型ロボットを操作する

Java NAOqi SDKによるプログラミングでロボットを走らせる

Aldebaran (SoftBankグループ) はプログラミング可能な人型ロボットを設計しており、ここ10年足らずで、NAO、Pepper、Romeoという3つのコンパニオン・ロボットを開発しています。

この3つの人型ロボットでは、同じオペレーティング・システムが動作しています。GentooベースのGNU/LinuxディストリビューションであるNAOqiです。NAOqiはAldebaranのロボットを動かす主要なソフトウェアであり、その役割に必要なすべてのライブラリとプラグ

ラムがこのディストリビューションに含まれています。つまり、NAOqiは分散ロボット・アプリケーションを作成するためのフレームワークであるとも言えます。一般的に、このようなアプリケーションはPythonやC++で記述されますが、[libqi](#)ライブラリを経由することで別のプログラミング言語を使用することも可能です。このライブラリはクロス・プラットフォームであり複数言語をサポートしているため、NAOqiも、別のプラットフォーム (Windows、Linux、Mac OS) や任意の言語 (対応す

るバインディングが存在するもの)で使用できます。

NAOqiを簡単に使えるように、Javaバインディングと専用のラッパー・クラスを提供しているのがJava NAOqi SDKです。このSDKはJARファイル形式で、他のサード・パーティ製Javaライブラリと同じように使用できます。

Java NAOqi SDK

この記事では、Java NAOqi SDKを使用して新しいロボット・クライアント・アプリケーションを作成する方法を説明します。ロボットは、右や左に曲がる、前進する、速く歩く、ゆっくり歩く、停止するなどの命令を聞き、そのとおりに動きます。

それでは、まず[アルデバランコミュニティ](#)のWebサイトからJARファイルをダウンロードしましょう(JARファイルは、お客様専用となっておりますので、ご注意ください)。プラットフォームとNAOqiのバージョンごとに、JARファイルが1つ存在します。この記事のサンプルは、NAOqiバージョン2.1.4を使用


し、Linux x64とMac OS Xでテストしています。お使いのプラットフォームに対応するJARファイルをダウンロードして、ビルド・パスに追加してください。

最初の手順は、**リスト1**に示すように、**main**メソッドで**Application**を作成することです。その際、**Application**のコンストラクタの引数にロボットのIPアドレスを渡す必要があります。**Application**は、フレームワークを初期化して**Session**に接続する役割を持っています。**Session**は、ローカルまたはネットワーク上のサービスを束ねるものです。**Application**を開始すると、**Session**が作成され、ロボットに接続されます。

Sessionの準備が整うと、サービスが取得できます。サービスとは言わば、ロボットが実行できる動作を公開するインタフェースです。ロボットにはそれぞれServiceDirectoryがあり、使用できるサービスの一覧が提供されています。この例では、**リスト2**に示す次のサービスが必要になります。

- `ALSpeechRecognition`: ☐



 ボイス・コマンドに応答するロボット
(動画)

写真提供:ALDEBARAN

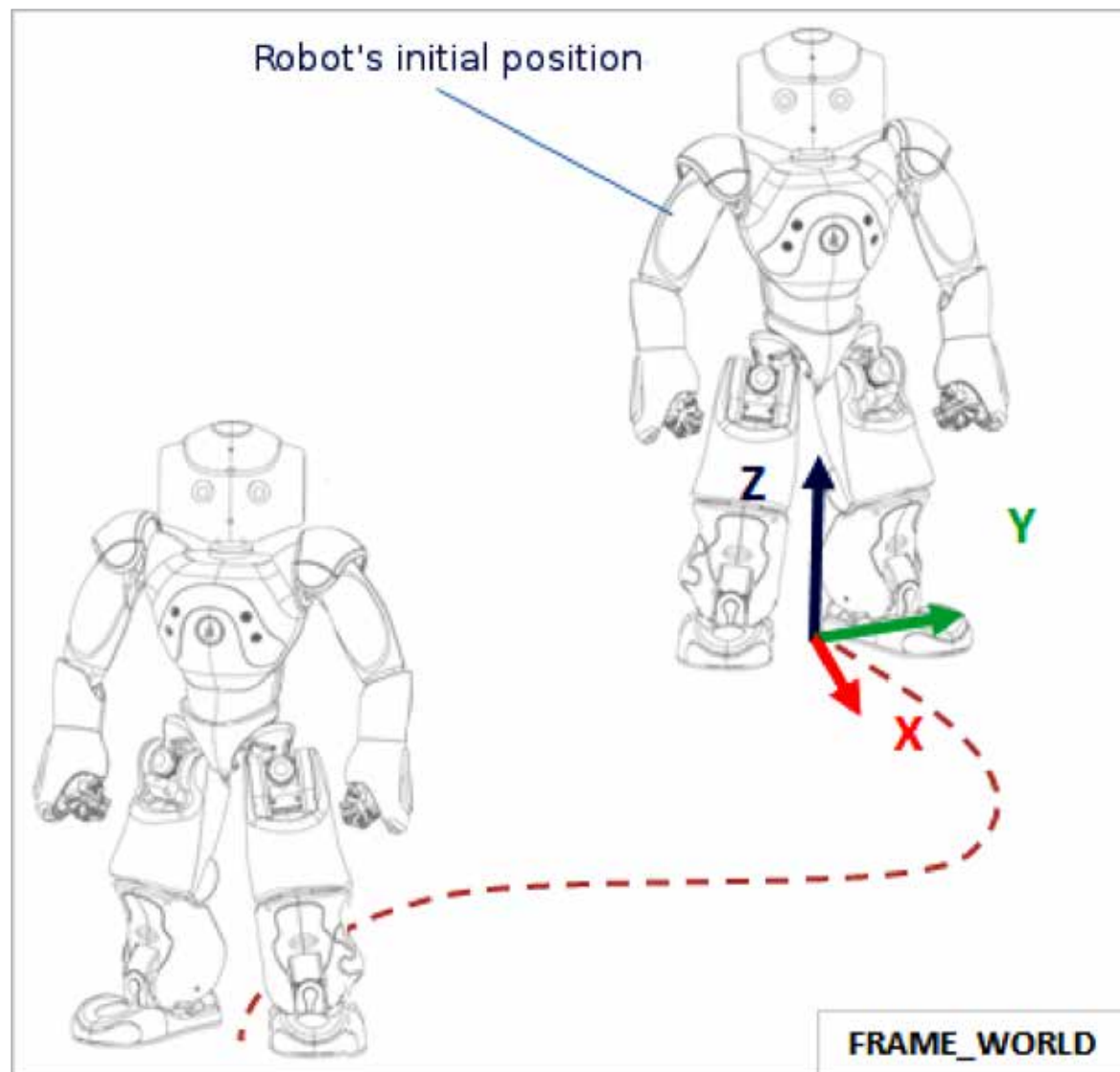


图1

ボットが、事前に定義された単語やフレーズを複数言語で認識できるようにするサービスです。

- **ALMemory**:ハードウェア構成に関する重要な情報の格納や取得のため、集中型のメモリを提供するサービスです。イベント通知を配信する際のハブとしての役割も果たします。
- **ALMotion**:ロボットを動かすためのサービスです。
- **ALTextToSpeech**:ロボットに話を

させるためのサービスです。

- **ALAutonomousMoves**: ロボットが繊細な動きを自律的に行うようにするサービスです。

それではまず、**ALSpeechRecognition**から使ってみましょう。ロボットが命令を聞いて理解するためには、ボキャブラリの登録が必要です(**リスト3**)。この例では、ロボットに与える指示のリストがボキャブラリにあたります。今回行うのは、


リスト1 / リスト2 / リスト3 / リスト4 / リスト5 / リスト6

```
public static void main(String[] args) throws Exception {
    String robotUrl = "tcp://nao.local:9559";
    // Applicationの新規作成
    Application application =
        new Application(args, robotUrl);

    // セッションを作成し、ロボットに接続する
    application.start();

    // 作成したセッションを取得する
    Session session = application.session();

    ...
}
```

 すべてのリストのテキストをダウンロード

ロボットに対して明確に指示された言葉を認識させることです。それ以上でもそれ以下でもないのので、「word spotting」は無効にしておきます。

次に、リスト 4 のように **ALSpeechRecognition** サービスをサブスクライブし、ロボットが音声入力を受け付けるようにします。

ロボットが言葉を認識すると、ALMemory WordRecognizedイベントが発生します。このイベントには、認識された言葉と、その言葉が実際に話された言葉であるという確率の推定値が含まれています。この入力进行处理するために、**リスト5**のようにイベントをサブスクライブしてコールバックを設定します。イベントをサブスクライブすると、サブスクリプションIDが

返却されます。これは、後ほどサブスクライブを解除する際に使用します。

EventCallbackは関数インターフェースなので、**リスト6**のようにラムダ式を使うこともできます。

では、ロボットが言葉を認識した際に、その命令に応じて動くようにしてみましょう。ロボットを動かすには **ALMotion** サービスを使用しますが、次の軸定義を考慮する必要があります。**図1**のように、X軸はロボットの正面方向、Y軸は右から左への方向、Z軸は垂直方向が正になります。

ALMotionのmoveToward(float x, float y, float theta)メソッドを使用すると、ロボットを前進、左右への回転、加速/減速、停止させることができます(リスト7)。その際に、次のパラメー

タを指定します。

- **x**は浮動小数点型で、単位を持たず、X軸方向の速度を表します。+1と-1が、それぞれ前進、後退の最大速度に対応しています。
- **y**は浮動小数点型で、単位を持たず、Y軸方向の速度を表します。+1と-1が、それぞれ左、右方向の最大速度に対応しています。
- **theta**は浮動小数点型で、単位を持たず、Z軸を中心に回転する速度を表します。+1と-1が、それぞれ反時計回り、時計回りの最大速度に対応しています。

ロボットは命令を聞いている間も自律的に動作しています。そこで、ロボットが転ぶリスクを減らすため、「expressive listening」を無効にします。というのも、ロボットが自律的な動作をしながら命令に従おうとすると、両方の動作が重なって転んでしまうことがあるからです。具体的には、命令を聞き始める前に**リスト8**のコードを追加します。

この例では、ロボットの頭の触覚センサーに触れると、まだ命令を聞き始めていない場合は命令を聞き始め、そうでなければ命令を聞くのを終了するようにします。そのために、**MiddleTactilTouched** イベント (イベント名の Tactile のスペルは間違っています) もサブスクライブします。

リスト9を参照してください。

リスト10のように、ラムダ式でも同じことができます。

ALTextToSpeechサービスを使用すると、ロボットに自身の状態をフィードバックさせることができます。たとえば、**リスト11**の `disableListening()` メソッドのような使い方ができます。

以上で、このJavaアプリケーションのテストを始める準備がほぼ整いました。あとは、アプリケーションがサブスクライブしたイベントを受信し続けるようにするだけです。`main`メソッドに以下の行を追加して、アプリケーションが終了するまでメイン・スレッドをブロックします。

フレームワーク
NAOqi は、同種
プログラミング、
サービス間通信、
情報共有を可能
にする**強力な**
フレームワーク
です。

```
// アプリケーションの実行を  
// 継続する  
application.run();
```

なお、アプリケーションの実行を終了する際には、**リスト12**のようにすべてのイベントのサブスクライブを解除して、ロボットを「クリーン」な状態に戻す必要があります。

アプリケーションの実行


このアプリケーションの実行中に何が起こっているのか詳しく理解するために、Choregraphe Memory

リスト7 リスト8 /リスト9 /リスト10 /リスト11 /リスト12

```

public void handleOrder(String order) throws CallError,
    InterruptedException {
    float x = 0;
    float y = 0;
    float theta = 0;
    if (order.equals("forward")) {
        x = 0.6f;
    } else if (order.equals("left")) {
        theta = 0.4f;
    } else if (order.equals("right")) {
        theta = -0.4f;
    } else if (order.equals("stop")) {
        x = 0f;
        y = 0f;
        theta = 0f;
    } else if (order.equals("faster")) {
        if (x > 0)
            x += 0.2;
        else
            x -= 0.2;
    } else if (order.equals("slower")) {
        if (x < 0)
            x += 0.2;
        else
            x -= 0.2;
    }
    motion.moveToward(x, y, theta);
}

```

 すべてのリストのテキストをダウンロード

watcherとRobot viewを使用します。
このツールを使えば、ALMemoryの
イベント、ロボットが話す言葉、ロボッ
トの動きを監視できます。

1. 最初にロボットをChoregrapheに接続し、MiddleTactilTouchedイベントとWordRecognizedイ

ベントをMemory watcherに追加し、Robot viewを有効にします。次に、Javaアプリケーションを実行します。**図2**を参照してください。

- 2. ロボットの頭部中央の触覚センサーに触れます。**

MiddleTactilTouchedイベントが発生し、ロボットの目のLEDが紺色に変わります（この変化はRobot viewには表示されません）。これは、ロボットが命令を

待っている状態であることを意味します。

3. ロボットに向かって「forward」と言います。**図3**のように、ロボットが聞いた言葉とその確率を含む

WordRecognizedイベントが発生します。ロボットが前進します。

4. ロボットに向かって「left」と言います。leftを含むWordRecognizedイベントが発

生し、ロボットは左に曲がります
(図4)。

5. ボキャブラリに含まれる他の言葉「right」「faster」「slower」「stop」も試してみてください。

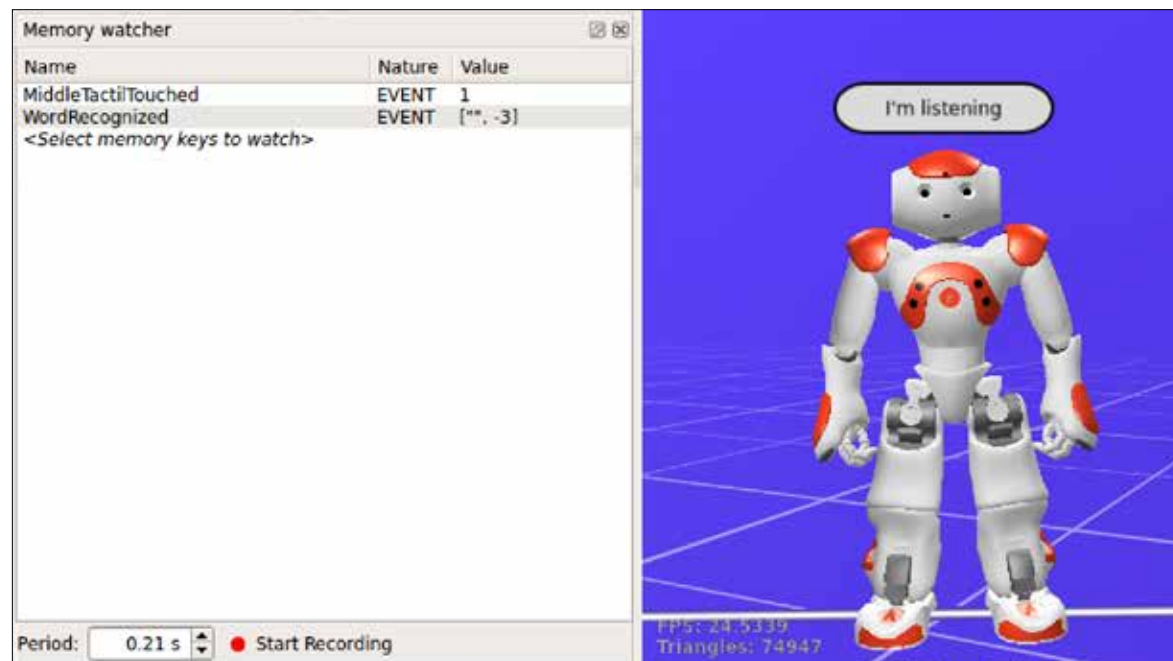


図2

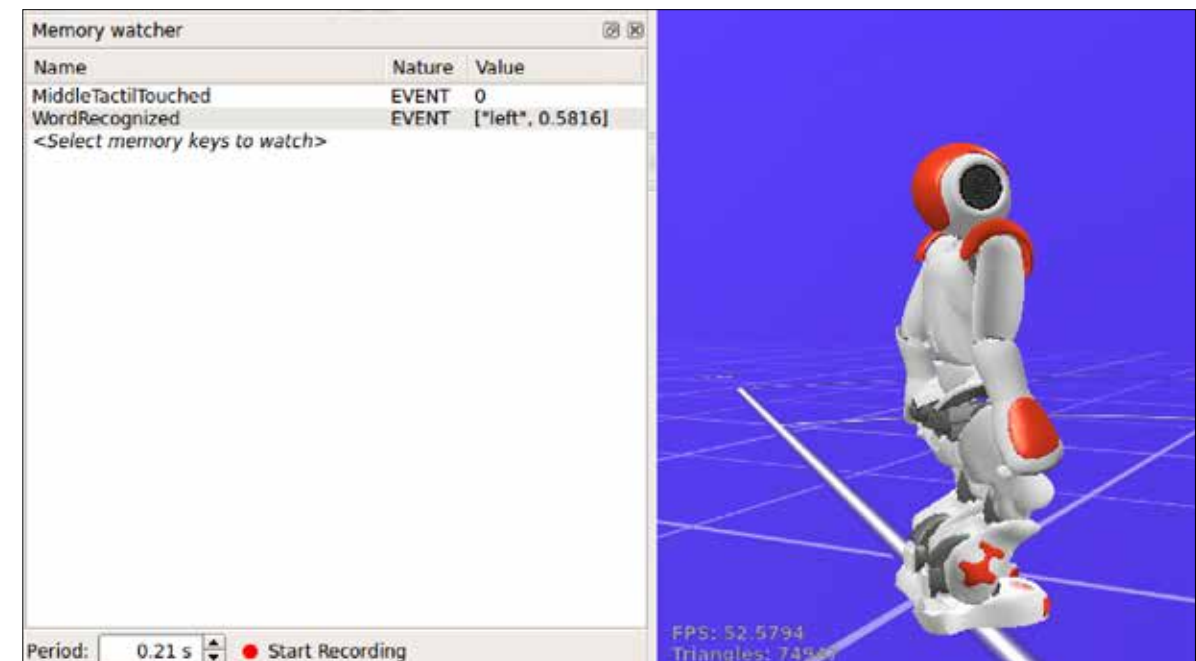


图4

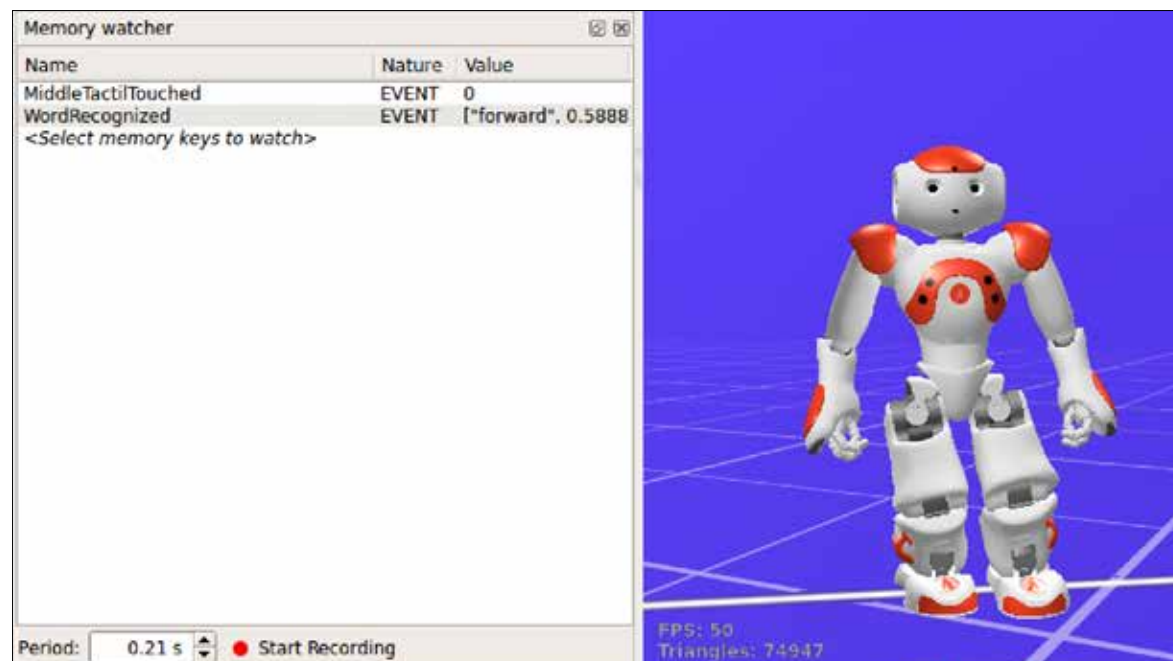


图3

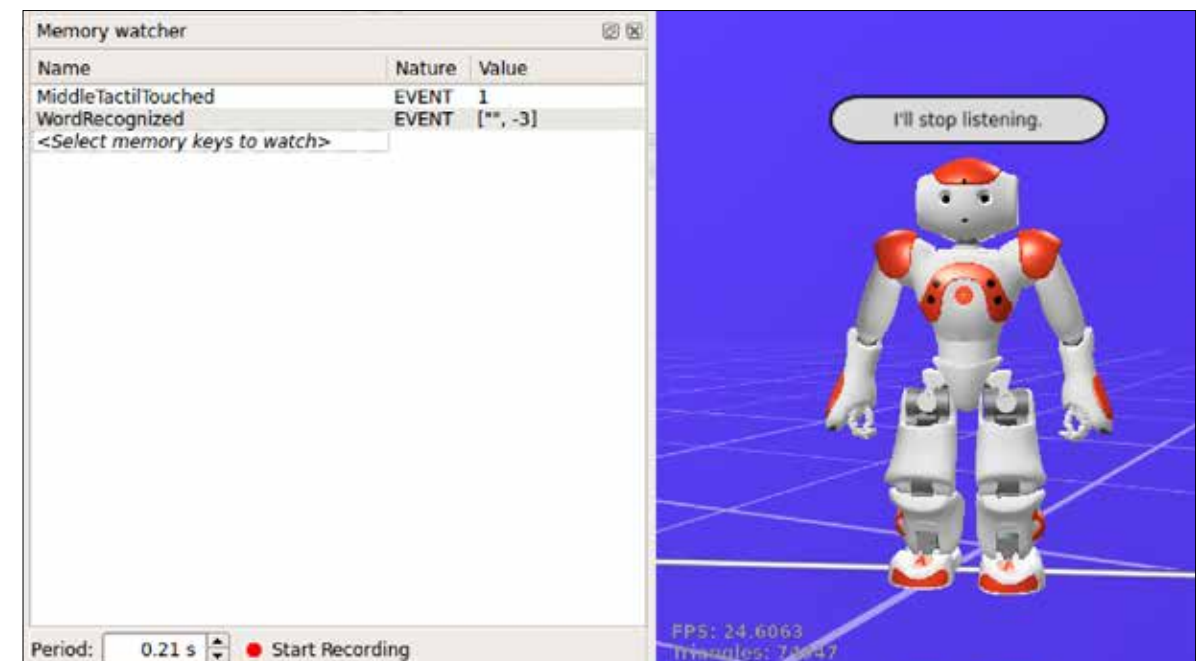


图5

[illegible]