

# Java™ magazine

By and for the Java community 

```
package agenda.rest;

@Path("/agenda")
public class AgendaResource {
    private volatile Agenda agenda;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Conference> listConferences() {
        return agenda.listConferences();
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public void addConference(Conference conference) {
        agenda.addConference(conference);
    }

    @GET
    @Path("/{id}")
    public Conference getConference(@PathParam("id") String id) {
        return agenda.getConference(id);
    }
}
```

ORACLE CLOUD

## イノベーションのための プラットフォーム

Java ベースのサービスが提供する柔軟なツールによって、今までにないタイプのアプリケーションを開発、導入できます。

## 01



//from the editor /

ツ



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

**ールがクラウドを作ります。**今日のクラウド開発環境は、最新型の開発手法に対応するために、ソフトウェア開発のライフサイクル全体を支えるものでなければなりません。クラウドで作業する開発者は、ビルド、継続的インテグレーション、ソース管理、チーム・コラボレーションのためのツールを必要としています。それらのツールがすぐに使える状態にあれば、クラウドをイノベーションのためのプラットフォームとして存分に活用できます。



Oracle Cloud Application Foundation 製品管理担当バイス・プレジデントの [Mike Lehmann](#) へのインタビューでは、オラクルが開発者向けに提供している Java ベースのクラウド・サービスの内容とその利点について聞きました。「ビジネスの新しいアイデアを後押しするようなアプリケーションの構築を、以前よりもはるかに短い期間で、しかも低いリスクで行うことができます」と Lehmann は語ります。このインタビューでは、Oracle Java Cloud Service、Oracle Developer Cloud Service などの、開発者にエンド・ツー・エンドの Java 開発 / デプロイ環境を提供する各種サービスについて、Lehmann の見解を知ることができます。

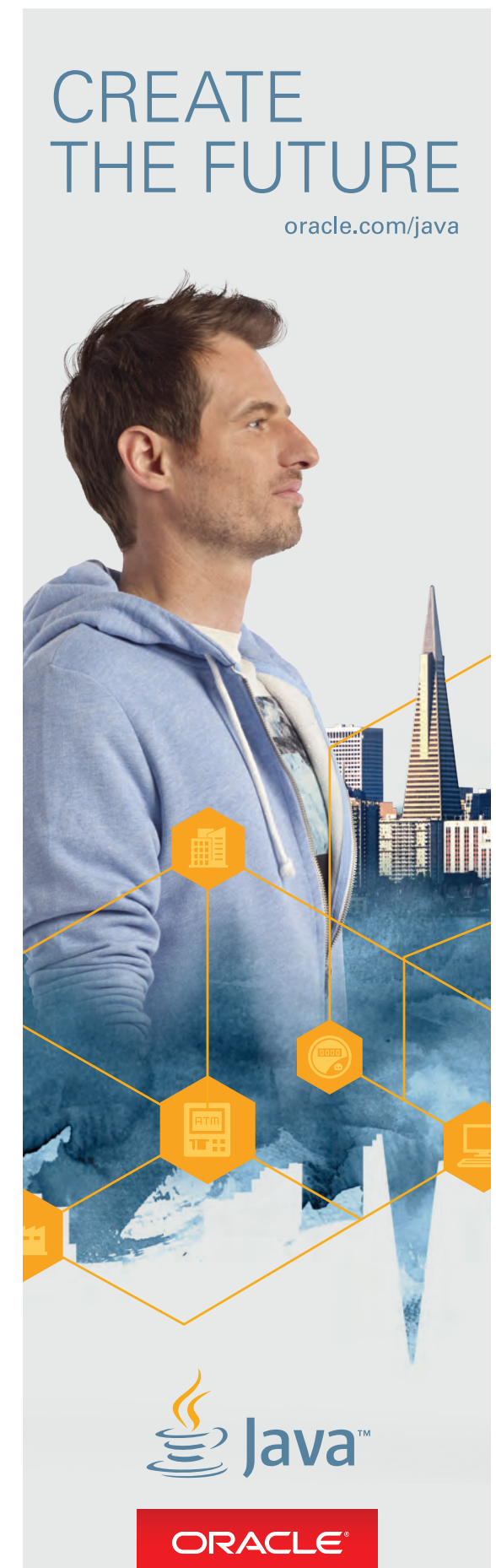
オラクルのクラウド・サービスを試したくなったら、Hardshad Oak 氏の記事「[Oracle Developer Cloud Service 入門](#)」をぜひお読みください。さらに、[Bert Ertman](#) 氏は、モジュール式クラウド・アプリケーションを Java で構築する方法を説明しています。

今も昔も変わらず、新年になると誰もが自分磨きを考えます。良い職に就き交流の幅を広げたい方には、[Bruno Souza](#) 氏と [Edson Yanaga](#) 氏のプランがぴったりです。彼らの記事では、コード、コミュニティ、そして本号のテーマであるクラウドについて語られています。この記事に記されているアクションは、将来の可能性を広げるのに役立ちます。

最後になりますが、Java Magazine 編集長としての私の役目は本号で終わりです。Java Magazine を熱心に支えてくださった読者の皆さんに感謝いたします。2015 年の Java Magazine も素晴らしいコンテンツを多数ご用意しておりますので、引き続きご購入ください（そしてコーディングも継続しましょう）。

編集長 Caroline Kvitka

写真：BOB ADLER



COMMUNITY

JAVA IN ACTION

JAVA TECH

ABOUT US

O

f

Twitter icon

java.net

blog

Java logo

02



# イノベーションのためのプラットフォーム

Oracle Cloud プラットフォームの Java ベースのサービスが提供する柔軟なツールによって、今までにないタイプのアプリケーションを開発、デプロイできます。**CAROLINE KVIKA**



クラウド・コンピューティングによって開発者のチャンスは拡大し続け、提供されるサービスも増加しており、そういった傾向は特に簡易性、生産性、イノベーションの面で顕著です。開発者にとって Oracle Cloud プラットフォームのどこが新しいのかについて、Oracle Cloud Application Foundation 製品管理担当バイス・プレジデントである Mike Lehmann にインタビューを行いました。

**Java Magazine** : Java 開発者にとって、クラウド・コンピューティングの台頭は、

イノベーション、コスト削減、生産性の観点でどのような意味を持っていますか。

**Lehmann** : クラウドの登場は、開発者や管理者によるソフトウェア、ハードウェアの扱い方を変えつつあります。実際、今の時点でも、とても簡単な方法で、クラウドの処理能力とエンタープライズ・ソフトウェアをすぐに利用できます。その結果、開発者や企業のアプリケーション構築の方程式が変わります。ビジネスの新しいアイデアを後押しするようなアプリケーションの構築を、以前よりもはるかに短い期間で、しかも低いリスクで行うことができます。

また、クラウドは、開発者によるアプリ



Oracle Cloud Application Foundation 製品管理担当バイス・プレジデント、Mike Lehmann





オラクルの製品管理担当ディレクターである Anand Kothari とともに、クラウドのアーキテクチャに関するドキュメントを見る Lehmann。

ケーションのアーキテクチャ設計の方法も変えようとしています。たとえば、フロントエンドにモバイルを利用する Java アプリケーションを構築し、そのパフォーマンス向上のためにキャッシュを使い、キューイング・サービスによって統合し、そして永続化のために標準的な SQL バックエンドを配備するとしましょう。この組み合わせは、Oracle Cloud で現在すでに提供しているサービスそのものであり、開発者はこれらのサービスをニーズに合わせて使うことができます。こうしたサービスの利用こそ次世代型のパラダイムであり、これまで私たちがオンプレミスで、ミドルウェアと呼ばれるものによって構築したシステムを置き換えるもの

の起点となるのが Oracle Java Cloud Service です。このサービスによって、開発者は数分で Java EE サーバーをプロビジョニングできます。その Java EE サーバーは、ユーザーが定義した、Java インフラストラクチャのクラスタ環境となります。この環境には、バックアップ、リカバリ、パッチ適用、スケールイン/スケールアウトを実行できる、ライフサイクル管理用のクラウドツールが付属しています。基盤となるインフラストラクチャを構成するのが Oracle WebLogic Server であり、市場をリードする標準ベースのアプリケーション・サーバーです。そして、永続化のために Oracle Database Cloud Service に直接プラグインできます。

### イノベーションのための柔軟性

「本当の意味でエンド・ツー・エンドの Java 開発 / デプロイ環境を得て、その高い柔軟性と多くのチャンスを活用して画期的なアプリケーションを構築できます」

です。Oracle Cloud を使うことで、開発者はプラットフォーム・インフラストラクチャの選定やそのインストールとセットアップを行わずに、構築とデプロイの作業に集中できます。

**Java Magazine**：開発者が利用できる Oracle Cloud のサービスにはどのようなものがありますか。

**Lehmann**：オラクルが提供する Java ベースのクラウド・サービス

さらに、Oracle Java Cloud Service と連携して動作する Oracle Developer Cloud Service が、開発ライフサイクル全体のプラットフォームとして機能します。ソース・コード管理による継続的インテグレーションをサポートするため、このサービスは、Git リポジトリと Hudson ビルド・システムを搭載しています。また、Eclipse や Oracle JDeveloper などの標準の開発ツールと統合され、組込みの Wiki、問題追跡、チーム管理といったコラボレーション・ツールも提供しています。

さらに、Oracle Java Cloud Service を補う各種コア・プラットフォーム・サービスがあります。

まず、Oracle Messaging Cloud Service では、JMS (Java Message Service) のキューとトピックを基にしたメッセージング・アプリケーションをクラウド内に構築できます。クライアント・アプリケーションは、ネイティブの Java API に加えて、REST API 経由でもこのサービスを利用できます。

次に、Oracle Coherence のキャッシュ機能により、Java オブジェクトをメモリ内にキャッシュできます。この機能は、Oracle WebLogic Server の Managed Coherence Server とい



最新の考え方  
「(クラウドは) 間違  
いなく、アプリケー  
ションの構築方法  
を変えています。  
統合されたチーム  
環境で、テスト環  
境を驚くほど短時  
間でプロビジョニ  
ングできるからで  
す」

う機能を利用しており、Oracle Java Cloud Service の新しい拡張機能として 2015 年にリリースされる予定です。

また、2015 年には、クラウドの Java 仮想マシン (JVM) として、Oracle Java SE Cloud Service もリリースされます。標準的な JVM ベースのアプリケーションを構築したいという多くのお客様からのご要望に対応した形です。Spring などの人気の Java フレームワーク、あるいは、Play などの今新たに人気が出ている Java ベース・スクリプト言語を利用しているお客様にも適しているのではないのでしょうか。Oracle Java SE Cloud Service は、Oracle Java Cloud Service と同じように、Oracle Developer Cloud Service、Oracle Java Cloud Service、Oracle Database Cloud Service などのさまざまなサービスで構成される巨大な Oracle Cloud エコシステムと直接統合されます。

オラクルは、Node.js などのサーバーサイドの JavaScript ソリューションの台頭についても十分に把握しています。現在は Oracle Java SE Cloud Service と並行して、2015 年のリリースに向けて、Oracle Node Cloud Service を開発しています。このリリースによって、Oracle Cloud に非常に優れた JavaScript サーバーが導入され、さらに Oracle Developer Cloud Service による JavaScript 開発ソリューションも加わることになります。そうなれば、JavaScript 関連のソリューションを Java クラウド・サービスと組み合わせ、データベース・クラウド・サービスと統合した上で簡単に構築、デプロイできるようになります。

このように、モビリティ、統合、ドキュメント、ビジネス・インテリジェンスのためにわざわざ高レベルなサービスを導入しなくても、本当の

意味でエンド・ツー・エンドの Java 開発 / デプロイ環境を得て、その高い柔軟性と多くのチャンスを活用して画期的なアプリケーションを構築できます。

**Java Magazine** : オラクルは、開発者が求める手法を実現するため、必要となるすべての機能を提供しようとしているのでしょうか。

**Lehmann** : オラクルは、Oracle Developer Cloud Service を通して、アプリケーションの

ライフサイクル全体を管理するインフラストラクチャを構築し、「どうやってビルドするか」、「どうやってソース・コードを管理するか」、そして「どうやって開発チームを管理するか」という疑問に答えようとしています。デプロイ側の視点で言うと、オラクルは Java クラウド・サービスによって、アプリケーションに一般的に備わっている非常に充実したマルチティア・トポロジを提供し、しかもほぼ透過的にそれを行って



オラクルのソフトウェア開発担当ディレクターである Nilesh Junnarkar とロードバランサのプロビジョニングについて話す Lehmann。





の視点で言うと、チーム開発や継続的インテグレーションと、本格的な本番品質のデプロイメント・インフラストラクチャの両面を実現できます。

マルチティアのシナリオでもよく使われる典型的な例として、Java アプリケーション・レイヤーの前方に別のレイヤーを置くことができます。その前方のレイヤーではおそらく Node.js のフロントエンドを使い、フロントエンドの HTML5 アプリケーション向けに、JSON オブジェクトの一部の形式を整えることになります。アーキテクチャの視点から言えば、このようなマルチティア・アーキテクチャを、それほど苦勞せずに面白い方法でまとめることができます。以前であれば、このようなアーキテクチャの構築には、IT 環境の複雑さに応じて数週間、あるいは数か月かかったでしょうが、今や基本的にオンデマンドです。そのため、アプリケーションに対する考え方や、アプリケーションのアーキテクチャ設計が根本的に変わります。オラクルによるエンタープライズ品質のソリューションを最初からすぐに利用できるのですから。

**Java Magazine**：これらのクラウド・サービスは、最新型の開発手法をどのように支援しますか。

**Lehmann:** Hudson などのビルド・インテグレーション・システムを、Maven による依存性管理や Git によるソース・コード管理と一緒に利用するというのが、近頃の開発における新しいトレンドとスタイルになってきています。このような方法によって、継続的インテグレーション・ソリューションを最初からすぐに利用でき、継

続的デリバリ・モデルさえも検討できます。

クラウド導入後に以前と少し変わる点は、以前は自分たちでこの環境をセットアップしなければならず、そのセットアップが複雑だったことです。しかし、クラウドではその環境が最初からすぐに手に入ります。クラウドは間違いなく、アプリケーションの構築方法を変えています。統合されたチーム環境で、テスト環境を驚くほど短時間でプロビジョニングできるからです。いや、テスト環境でプロビジョニング、実際のテスト、後処理までを非常に短時間でやるケースの方が多いでしょうか。これらの機能が環境の中にそのまま組み込まれているので、セットアップのために多大な投資を行う状態と比較すれば、その違いは明らかです。これが本当の最新型の開発パラダイムと言えます。

**Java Magazine**：パブリック・クラウドへの移行に対するユーザーの抵抗感は最近になって薄れてきているのでしょうか。

**Lehmann:**パブリック・クラウドかプライベート・クラウドかという議論がしばらく続いてきましたが、私の感覚では、このような議論は徐々になくなると思います。オラクルのお客様とクラウドについてお話すると、本番のワークロードはオンプレミスで維持しながらも、開発やテストをクラウドに移行することには抵抗を感じないユーザーがすでに増えていることがわかります。しかし、これはまだ、クラウドへの転換のほんの一部だと思います。2年前には、パブリック・クラウドで何もしようとしなかった人たちが、今は開発やテストの段階に進んでいるのですから。数年後、パブリック・クラウドが優れたセキュリティとスケーラビリティを







# JCPの選挙結果

## 2014年秋のJava Community Process (JCP) Executive Committeeの選挙

**結果:**この選挙はVotenet上で行われ、2014年11月10日午後11時59分(太平洋標準時)に投票が締め切られました。

この年の選挙では、8つの批准議席と5つの選出議席について投票を行いました。JCPは2015年にJCP 2.10へと移行するため、今回の議席獲得者は1年の任期を務めることになります。

批准議席を獲得(または再獲得)したJCPメンバーは、Freescale、Gemalto M2M GmbH、Goldman Sachs、MicroDoc、SAP、Software AG、TOTVS、V2COMです。

選出議席を獲得(または再獲得)したJCPメンバーは、ARM、Azul Systems、Hazelcast、Werner Keil氏、Geir Magnusson, Jr.氏です。

新たに選出されたメンバーは、2014年11月25日に議席につきました。

詳しい選挙結果については[こちら](#)をご覧ください。

写真提供: BEJUG

今注目のJAVA SPECIFICATION REQUEST

# JSR 364: BROADENING JCP MEMBERSHIP



## JSR 364: Broadening JCP Membership

は、JCP.nextによる取り組みの第4の要素で、2011年5月にJava Community Process (JCP) チームによって開始されました。JCP.nextは、JCP内部の透明性、参加状況、機敏性、ガバナンスを改善することを目指しています。JSR 364の具体的な目標は、「新しいメンバーシップ・クラス

の定義、既存のメンバーシップ・カテゴリの変更、コミュニティからの参加の実現、およびJCPメンバーからの、知的財産に関する適切な合意の取得によって、より広い範囲からJCPへの参加を促すこと」です。

**Heather VanCura氏** (写真) がJSR 364の仕様リードです。この専門家グループには、London Java Community (LJC) とSouJavaの代表者や、大小の企業の代表者が集まっています。参加者全員が、より幅広いJava開発者コミュニティとJCPの関わりを改善し、交流を深めることを目指しています。

JSR 364の進捗状況をご覧になる場合や、この取り組みへのご意見や新しいご提案がある場合は、[Java.net](#)のJSR 364プロジェクトにアクセスしてください。このページでは、JSR 364に皆さんの意見を反映するための各種の手段を取り揃えています。







## SAFE WATER KENYA

(Safe Water Team  
によるプロジェクト)  
[safewaterteam.org](http://safewaterteam.org)

### 本社所在地:

ミシガン州グランドラ  
ピッツ

### 業界:

非政府組織 (NGO)

### 使用している Oracle テクノロジー:

JDK、Oracle Database  
12c、Oracle  
WebLogic Server  
12c、Oracle Database  
Mobile Server、  
Oracle Berkeley DB  
Transactional Data  
Store



mFrontiers の  
Daniel Pahng 氏  
(右) が開発した  
最新の Survey  
App を確認す  
る、Safe Water  
Kenya の Don  
Arnold 氏 (左)

### 大きな課題

Water Project に  
よると、清潔で安  
全な飲料水を手  
できない人々は 10  
億人以上に上りま  
す。

Kenya (SWK) プロジェクトの発起人兼  
エグゼクティブ・ディレクターである Don  
Arnold 氏です。ミシガン州グランドラピ  
ッツを拠点とする非営利組織 SWT は、発展  
途上国での水質の改善に取り組んでいま  
す。

「アフリカでは定期的に給与を得ている  
人々はそれほど多くありません」、衛生設備  
関連の特許を 30 以上持つ経験豊かな水  
道設備コンサルタントである Arnold 氏は  
言います。「つまり、病気になって働けない

日があれば、その日は収入がありません。  
子供が病気になれば、母親は家で子供の  
面倒を見る必要があるため、お金を稼げず、  
結果的に子供も学校に行けなくなります」

東アフリカにあるケニア共和国の農村  
部で発生しているこうした重大な飲料水  
問題に対処するため、SWK は高度な技術  
を必要としない緩速砂ろ過法を使用した  
[Hydraid BioSand Water Filters](#) を 1 年以  
上前に設置しました。「これまでに設置し  
た (ろ過器の) 数は 2,500 に上ります」と

Arnold 氏は話します。「家族の人数は平均  
で約 7 名と考えられるため、わずか 1 年ほ  
どで 17,000 人の暮らしに影響を与えてきた  
ことになります」

ろ過システムは無料ではありません。「援  
助提供者に対して設置記録を提供するた  
め、写真や GPS 座標、受領者のサインを  
含む詳細な調査票に入力する必要があります」と Arnold 氏は話します。

SWK はまさにハイテクなソリューション  
を使用してこれらの要件を満たしています。

写真: ANDREA MANDEL





## Arnold氏（赤色のシャツ）とSafe Water Kenyaの倉庫スタッフおよび設置チーム・メンバー

すべて Java で構築された "Survey App" は、イリノイ州リバティビルを拠点とするエンタープライズ・ソフトウェア開発企業、オラクルのパートナーでもある [mFrontiers](#) によって SWK 専用開発され、携帯型 Android タブレット上で稼働しています。mFrontiers はこのアプリのおかげで、2014 Oracle Excellence Award for Sustainability Innovations を受賞しました。

## 炭素から浄水へ

Arnold 氏が SWK を立ち上げようと思い立ったのは、6 年前、教会の任務でアフリカに派遣されていたときでした。「ウガンダで地元の指導者とともに車に乗って

いたとき、彼が道路沿いに車を停めて川の方を指さしたのです。そこでは、人々がプラスチック容器に茶色く汚れた水を汲んでいました」と、Arnold 氏は次のように当時を振り返ります。「人々はその水を家に持ち帰って、飲んだり料理に使ったりして病気になると言うのです。多くの場合、水を媒介とする病がどこから来たのか人々は分かっていないということでした」

SWKは手始めに2,250個のHydraidろ過器をケニアに送り、その設置に1年半かかるだろうと予想していました。しかし、農村の診療所と協力して貧しい世帯を探し出すSWKの現地マネージャー、Vivian Akinyi氏をリーダーとする地道で精力的

な活動のおかげで、SWK は 9 か月ですべてを設置し、その後も設置数を増やし続けています。「通常、1 日に 15 個のろ過器を設置しています」と Arnold 氏と言います。

当初、飲料水ろ過器およびその設置を賄うおもな資金として、同じくジュネーブを本拠とする非営利組織 Gold Standard Foundation が認定した炭素クレジットを利用していました。「現在のクレジットは、私たちが二酸化炭素の排出を減らすテクノロジーや機器を使用するという前提で認定されたものです」と Arnold 氏はこう説明します。「SWK のケースでは、安全に水が飲めるように、木材や木炭を燃やして水を沸かすという行為が減少します。水を沸騰させていた家庭にろ過器を設置するたびに、その分の排出量が削減されます。」

SWK の調査票はもともと、紙とペン、カメラ、GPS 機器を使用して現場で記入したものを、SWK の配送事務所に戻ってからコンピュータ・システムに再入力して、米国の SWK 本部に転送していました。しかし、この方法には時間がかかり、エラーが生じやすくなります。現在、SWK Survey App はオフライン・モードの Android タブレット上にデータを収集して保存し、手順を自動化してエラーを軽減しています。mFrontiers が開発した Survey App は完全に Java で構築されており、同社の mFinity エンタープライズ・モビリティ管理プラットフォームを利用して、すべて Oracle 製品からなるスタック上にデプロイされています。

ろ過器の設置後、SWK のスタッフが調査票に入力します。「調査票は Android タブレットで7～8 ページにわたり、各ペー



## ハイテクなしで浄水を

Safe Water Kenya (SWK) が使用する Hydraid BioSand Water Filter は持続可能なローテク装置であり、飲用、料理、公衆衛生向けに清潔で安全な水を提供することを目的としています。アルバータ州（カナダ）にあるカルガリー大学の David Manz 教授が開発した Hydraid ろ過器は、もともと 1800 年代初頭にスコットランドで開発され、1829 年に初の公共浄水施設をロンドンにもたらした緩速砂ろ過法をベースに生産されています。

緩速砂ろ過法は自然発生の生物学的プロセスを利用して水を浄化しており、砂の表面数 mm に自然増殖するゼラチン層または生物膜が、湖や小川から雨水までのあらゆる水源から不純物を取り除きます。「善玉菌が悪玉菌を食べてくれるのです」と説明するのは SWK の発起人兼エグゼクティブ・ディレクターの Don Arnold 氏です。

「浄水を提供するためのテクノロジーは多数ありますが、Hydraid ろ過器ほど実用的なものはほとんどありません。カートリッジなどの部品を交換したり、塩素などの化学薬品を追加したりする必要がないのです」と Arnold 氏は続けます。「Hydraid ろ過器は水に含まれる細菌、ウイルス、寄生生物を 95% 除去します。寄生生物の除去は塩素でも不可能です」

Manz 教授が緩速砂ろ過器に対して行った第 2 の刷新は、古くから使用されてきた大きくて重いコンクリート容器を、設置しやすい軽量の FDA 認可プラスチック容器で置き換えたことです。この容器は高さ 30.5 インチ、直径 16.5 インチで、空のときの重量が 8 パウンド、一杯にすると 140 パウンドです。30 分で設置でき、1 時間あたり最大 47 リットルの水をろ過します。これは 8 ～ 10 人家族には十分な量です。生物層は最初の水を上部に注いだ時点で増殖し始め、完全に形成されるまでに約 2 週間かかります。ごくわずかなメンテナンス（上部にたまった固形物を時々取り除くだけ）で、電気を使わずに清潔で安全な水を 10 年間提供します。



協力先の診療所を訪ねる Arnold 氏と Safe Water Kenya の現地マネージャー、Vivian Akinyi 氏（左）

ジに 5 ～ 6 の質問が含まれています」と語る mFrontiers のプレジデント、Daniel Pahng 氏は、クライアント側アプリ向けのオープンソース JavaScript ライブラリである jQuery を使用して、自ら Survey App を開発しました。「住所がないため、(SWK のスタッフは) タブレットを使って家族の写真を撮り、GPS 座標を記録します」

最後に、設置者は受け取り主のサインと Hydraid ろ過器のシリアル番号をタブレット・フォームに記録して、次の家庭に移動します。

これらの遠隔地にはインターネットが接続されていないため、Android タブレット上の軽量データ・ストア、Oracle Berkeley DB にデータが保存されます。Pahng 氏は説明します。「配送オフィスに戻ると、Android タブレット上で稼働している Oracle Database Mobile Server の同期エージェントが、クラウドベースの Oracle WebLogic Server 上にホストされている Oracle データベースに対して自動的に調査票をアップロードします。Safe Water Kenya の本部では、ブラウザを使用してここにある情報にアクセスできます」





左：砂と砂利を使用した HydrAid BioSand Water Filter の設置  
右：Safe Water Kenya のトレーニングを終えた公衆衛生従事者に対する卒業証書授与式

写真提供:SAFE WATER KENYA



Pahng 氏によると、mFinity プラットフォームは当初、Microsoft .NET で構築されていました。「しかし、グローバル展開を決めたとき、Microsoft はモバイル分野のリーダーではないことに気づいたのです」と Pahng 氏はこう続けます。「ほとんどのエンタープライズ・レベルの顧客にとって必要なのは Java でした」

## 過去に例のない透明性

Survey App は設置プロセスを迅速化するだけでなく、非営利組織では匹敵するものがほとんどないレベルの透明性を実

現します。SWK の Arnold 氏は次のように述べます。「多くの非営利組織は概要レベル以上の情報を援助提供者に提供していません。言い換えると、私たちのようにろ過器を設置しているなら、低頻度のニュースレターかおそらくは年次報告書で『昨年は X 個のろ過器を設置しました』と謹んで報告する程度でしょう」

「それでも構いませんが、私たちが提供するレベルには及びません」と Arnold 氏は続けます。「ろ過器を設置したその日のうちに、調査票はシステムにアップロードされ、世界各地にいる SWK のス

タッフと援助提供者がその内容を確認できます。家族の写真からあらゆる情報までが表示されます。このようなツールを使っている組織をほかには知りません。

スタッフと援助提供者に素早く情報を提供できるだけでなく、アプリとタブレット・コンピュータのおかげで SWK の活動も迅速になることを Arnold 氏は期待しています。「私たちは活動規模について現実的であるよう努めていますが、今年の計画ではろ過器を約 5,000 個に倍増するつもりです」

「mFrontiers とオラクル  
には深く感謝しています」  
と Arnold 氏はいいます。  
「これ以上は望めないほど

のシステムです| </article>

MORE ON TOPIC:



**Philip J. Gill**、カリフォルニア州サンディエゴを活動拠点とするライター兼編集者。20 年以上にわたり Java テクノロジーの動向を追いつけている。





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、Worldwide Java User Groups Community の設立者。

**Edson Yanaga** : Produtec 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氏の写真: BOB ADLER/GETTY IMAGES

# キャリアを磨くための3ステップ

スキルアップとネットワーク拡大のための、より効果的なコーディング、コミュニティ作り、クラウドの探究

**成**功のためには時間をかけ努力する必要があることは誰もがわかっています。Malcolm Gladwell氏は著書『Outliers』で、かの有名な「一万時間の法則」を生み出しました。どのような取り組みにも、成功するには(少なくとも、Bill JoyやSteve Jobsのように大きな成功を遂げるには)ざっとそれくらいの時間をかけなければならない、というものです。しかし、10,000時間とは極端な数字です。私たちが望むのは、毎日少しずつキャリアを磨くことだけです。

何かを始めるのに、そんなに多くの時間は必要ありません。成功とは、たくさんの小さな、濃縮された努力の積み重ねなのです。Josh Kaufman氏は著書『The First 20 Hours』の中で、何かに20時間集中して努力すれば、驚くような成果が得られると述べています。20時間であれば悪くはないですね。これなら手を出せます。

開発者は、いくつかの大きな分野でスキルアップするだけで、キャリアアップすることができます。その分野とは、コーディング、コミュニティ、そしてクラウド・コンピューティングです。これら3つの分野

に20時間をかけることができれば、待望の出世を果たすことができます。家に帰って、この記事のエクササイズを試してください。驚くような成果が得られるはずです。

**コード: Code Kata**を導入する音楽、スポーツ、武道、プログラミングに共通することは何でしょうか。それは、偉大な業績は、練習を通じて成し遂げられるということです。理論、技法、ツールについて学ぶことも役に立ちますが、修得のためには練習とフィードバックが必要になります。

偉大な開発者になりたいければ、コーディングをしなければなりません。それも、大量に。もったもだと思いませんか。開発者としてステップアップするには、もっとコーディングしなければならない。実にシンプルです。コーディングは、開発者の仕事とほぼイコールです。ただ、現実には、開発者が毎日記述しているコードのほとんどが、自分を試すような難しいものではありません。しかし、自分を追い込まなければ向上しないのです。いつもと違う考え方を促さない単調なコードは、大量に記述

しても効果はありません。

武道では、指導法として「形」を利用します。練習生は、幾度となく形を繰り返すことで、自然な動きを学びます。プログラミングの世界で、Dave Thomas氏は「Code Kata」という言葉を作り出しました。Code Kataとは要

するに、反復作業の中でコードを拡張し洗練しながら、小さな問題を次々と解決していくことを言い表しています。その目的は、完璧な

別の選択肢への道  
「繰り返すことで、  
代替的なアプローチが明確になり、  
別の選択肢が開かれます」  
—Robert Genn

解決策を出すことではなく、練習を通じてプログラミング・スキルを磨くことです。

武道の原点にある思想の1つ、「守破離」は、アジャイル開発の世界で広まりました。その基本的な考え方として、知識は次の3段階を経て習得されます。

- 守: なぜそうするのかを疑わずに、ひたすら形に従う。
- 破: 上達すると、背後にある理論や技法を理解できる。他の情

```

1 import java.util.*;
2
3 public class ListTransformerTest {
4     private static final String[] strings =
5         {"a", "7", "4", "2", "1", "6", "10", "8", "3"};
6
7     private ListTransformer listTransformer;
8
9     @Before
10    public void setUp() {
11        this.listTransformer = ListTransformer.of(Arrays.asList(strings));
12    }
13
14    @Test
15    public void testGetSortedStrings() throws Exception {
16        assertEquals(Arrays.asList("10", "2", "4", "7", "1", "a", "6", "8", "3"));
17    }
18
19    @Test
20    public void testGetSortedIntegers() throws Exception {
21        assertEquals(Arrays.asList(2, 4, 7, 10));
22    }
23
24    @Test
25    public void testGetSortedDescendingIntegers() throws Exception {
26        assertEquals(Arrays.asList(10, 7, 4, 2));
27    }
28 }

```

Code Kata

Edson Yanaga氏がCode Kataの概念について紹介し、例を示しています。







のを作れるようになります。  
創造性を解き放つことができるのです。

クラウドの約束によって、「過去を繰り返す」というように）再現的に（reproductively）考えるのではなく、生産的に（productively）考えることができます。現実はそこまで甘くないでしょうが、そうであるかのように振る舞うことができます。

新しい思考法を受け入れるために、疑いの目を向けることを一時的にやめる練習をしましょう。クラウド・コンピューティングを試し、何もかもが無限になる環境でのコーディングに時間や労力をかけてみてください。ここで身に付けた見識は、職場での目の前の問題を解決するものではありませんが、自分のキャリアを面白い方向へと導いてくれるはずです。

**アクション:**クラウド・コンピューティング・プロバイダのアカウントを作成し、コードをデプロイする。

**アクション:**コンテナ、PaaS、ビッグ・データ、Hadoop クラスタなどの各種のクラウドを試す。

### まとめ

言うまでもなく、紹介したアクションを組み合わせれば、もっと楽しいことに発展します。Code Kata のセッションを JUG に提案するのはどうでしょうか。次回のミーティングでクラウドの経験について話すのも良いでしょう。あるいは、Code Kata とクラウド環境での解決策について、グループ・ディスカッションの場を

広くオープンに  
「新しい思考へと  
開かれた心は決  
して元の大きさに  
戻らない」

—Albert  
Einstein

提案してみてもいいがで  
うか。

今までとは違う考え方をし  
て、問題に対してとり得るす  
べての解決策を考察するた  
びに、開発者として成長しま  
す。それぞれの領域に 20 時  
間をかけて、そこから生ま  
れる大きな違いを実感して  
ください。

</article>

MORE ON TOPIC:



### LEARN MORE

- [Code Kata](#)
- [The Creativity Post](#)
- [JUG に参加](#)



## Find the Most Qualified Java Professionals for Your Company's Future

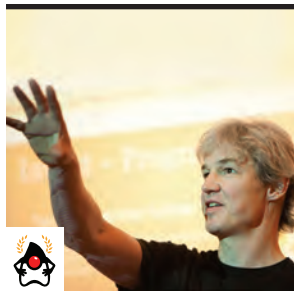
Introducing the *Java Magazine* Career Opportunities section – the ultimate technology recruitment resource.

Place your advertisement and gain immediate access to our audience of top IT professionals worldwide including: corporate and independent developers, IT managers, architects and product managers.

For more information or to place your recruitment ad or listing contact:

[tom.cometa@oracle.com](mailto:tom.cometa@oracle.com)





MICHAEL KÖLLING

**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。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 を研究分野としている。Java 教本2冊を出版。BlueJ および Greenfoot の主要開発者。

写真：JOHN BLYTHE

## パート2 Raspberry Pi での Java コーディング

BlueJ による Raspberry Pi ハードウェアへの対話型アクセス

**本**シリーズのパート1では、Raspberry Pi で BlueJ を実行し、2 台目のマシンを使用せずに、ポケット・サイズのコンピュータで Java プログラミングを直接行う方法について説明しました。デスクトップ・マシンで Java プログラムを開発して JAR ファイルを Raspberry Pi に送信する必要はもうありません。Raspberry Pi をデスクトップの代わりに使用して、Java を研究し、学習できる明るい未来がここにあります。

ただ、パート1では、デスクトップでも可能なことを Raspberry Pi で行っただけです。標準的な Java のサンプルと BlueJ を実行し、デスクトップやラップトップで提供されてきた機能を利用しました。

誤解しないでほしいのですが、標準的な Java プログラムの記述方法を学習するためだけに Raspberry Pi で BlueJ を使用することに問題があるわけではありません。Raspberry Pi と BlueJ の組合せは教育に適した素晴らしいツールです。そして、

Raspberry Pi を低価格で購入できる点が世界を大きく変えています。しかし、そこで終わる必要はありません。

次の楽しいステップは、以前は不可能であったことに取り組むことです。たとえば、Raspberry Pi の物理的なコンポーネントに Java から直接アクセスすれば、Java コードだけでなく、ハードウェアでもさまざまなことに挑戦できるようになります。容易にアクセスできる Raspberry Pi のハードウェアと、BlueJ の対話型の実験機能があれば、魅力的なことを実現できます。

本記事では、Raspberry Pi のさまざまなハードウェア・コンポーネントに直接対応し、簡単に使用できる Java オブジェクトの作成方法について見ていきます。

### 前提

本記事では、Raspberry Pi がセットアップされていること、ログインして X Window System サーバーを起動できる状態にあるこ

と、および BlueJ をすでにインストールして起動していることを前提とします（これらの前提を満たしていない場合は、先に本シリーズのパート1をお読みください）。

さらに、本記事のサンプルに従って作業を進めるためには、追加のアイテムとして、LED1 個、ボタン1個、抵抗1個、ワイヤー4本が必要になります(図1)。また、必須ではありませんが、ブレッドボード(実験用基板)があれば便利です。ブレッドボードを使用すれば、はんだ付けをせずに部品同士を簡単に接続することができます。ワイヤーの端をねじって部品を接続するだけでは、すぐにはずれてしまいます。

### BlueJ プロジェクトの取得

まず、BlueJ プロジェクトを開きます。RasPi-IO プロジェクトをこちらからダウンロードし、解凍して、Raspberry Pi 上の BlueJ で開きます(図2)。

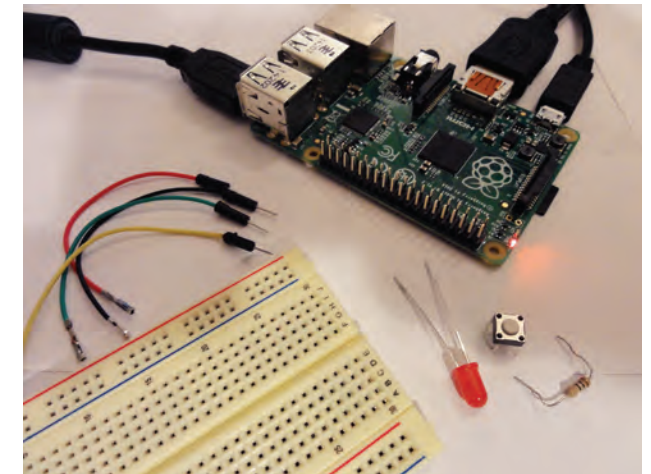


図1

このプロジェクトには、Raspberry Pi に接続可能な入出力部品を抽象化した2つのクラスが含まれています。1つは [GPOutput](#) クラスで、LED など、汎用入出力 (GPIO) ピンに接続できる任意の部品を表します。もう1つは [Button](#) クラスで、GPIO ピンに接続されたプッシュ式のボタンを表します。

### LED の準備

最初に取り上げる例は、LED を点灯または消灯するという非常に簡単なものです。

まず、お手元の LED を以下の手順で Raspberry Pi に接続



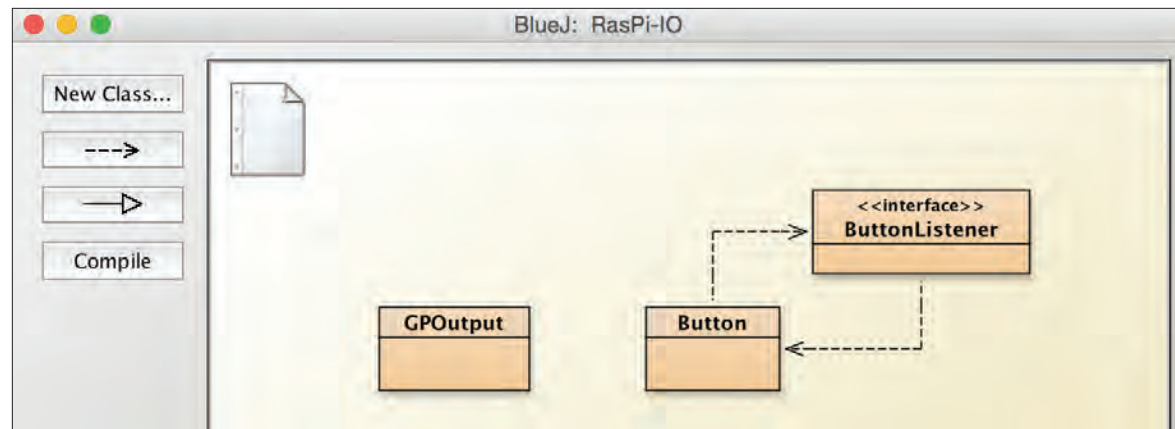


図2

します。

1. LED のアース端子 (短い方の脚) をワイヤーに接続します。
  2. このワイヤーの端を、Raspberry Pi の拡張ヘッダーのピン 6 (Ground) に接続します。Raspberry Pi のピン構成図を参考にしてください。
  3. LED の長い方の脚を抵抗に接続します。
  4. この抵抗の反対側の端を 2 本目のワイヤーに接続します。
  5. 2 本目のワイヤーの反対側の端を、Raspberry Pi 拡張ヘッダーの GPIO 1 (ピン 12) に接続します。
- 組立て後の全体の様子は図3のようになります。

注意点として、正しいピン番号体系を使用するようにしてください。現状ではさまざまなピン番号体系が利用されています。BlueJ は Pi4J ライブラリを使用するため、ここでは Pi4J ライブラリで用いられる番号体系に合わせます。かならず正しい番号体系を参照するようにしてください (先ほどの図を確認してください)。この番号体系で

は、GPIO 1 はピン 12 です。

### BlueJ での作業

現時点で LED は GPIO ピンのひとつに接続されています。次に、BlueJ で GPOutput クラスのオブジェクトを作成します (パート1で説明したとおり、BlueJ ではクラスを右クリックして、メニューでコンストラクタを選択することによって、オブジェクトを作成できます)。

表示されるコンストラクタには、使用する GPIO 番号を選択できるものと、デフォルトの GPIO 1 を使用するものの2種類あります。前項で LED を GPIO 1 に接続したので、ここではデフォルトのコンストラクタを使用できます。この操作により、オブジェクト・ベンチに、LED を表すオブジェクトが表示されます (通常は、このオブジェクトは GPIO 1 に接続されているあらゆる部品を表しますが、本記事では LED がその部品にあたります)。

以上で、オブジェクトとの対話によって LED を制御できるようになりました。本当に簡単です。gPOutput

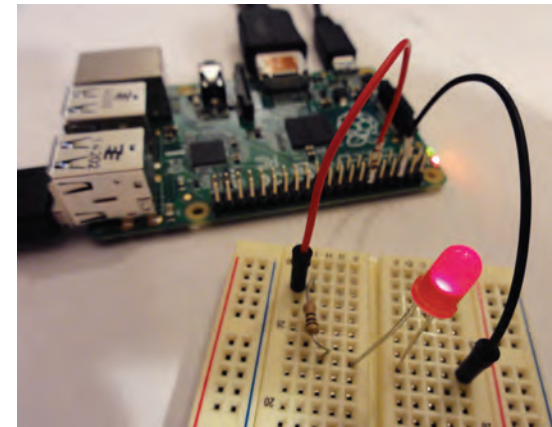


図3

オブジェクトを右クリックして on() メソッドを呼び出す (図4) ことによって、LED が点灯します。次に、LED の消灯を試してください。さらに、blink() メソッドも試すことができます。さまざまなパラメータ値を指定してこのメソッドを呼び出してみてください。

### ボタンの使用

次に、Raspberry Pi にボタンを接続し、Raspberry Pi の状態を読み取ります。ボタンを以下のように接続してください (図5)。

1. 3 本目のワイヤーを使用して、ボタンの一方のコネクタを、Raspberry Pi 拡張ヘッダーの 3.3V

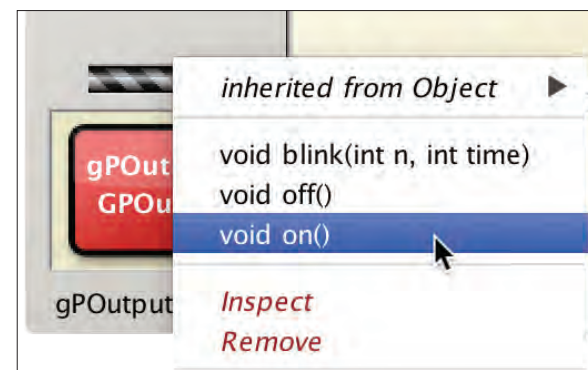


図4

- 電源 (ピン 1) に接続します。
2. 4 本目のワイヤーを使用して、ボタンのもう一方のコネクタを、Raspberry Pi 拡張ヘッダーの GPIO 4 (ピン 16) に接続します。

BlueJ で、Button 型のオブジェクトを作成して、このボタンを表すことができますので、実際に試してください。先ほどと同様に、デフォルトのコンストラクタと、使用する GPIO 番号を指定するためのパラメータを持つ第2コンストラクタがあります。デフォルトで使用される GPIO (この場合 GPIO 4) にボタンを接続したので、デフォルトのコンストラクタを使用すれば簡単にできます。

button オブジェクトには、ボタンの状態を確認するための2つのメソッド、isUp() と isDown() があります。ボタンに触らずに、つまりボタンが押されていない状態でこれらのメソッドを呼び出した場合、isUp() は true を返し、isDown() は false を返します。

ボタンを押したままの状態では、これらのメソッドを呼び出してみてください。button オブジェクトによって、ボタンの状態が正しく報告されるはずで

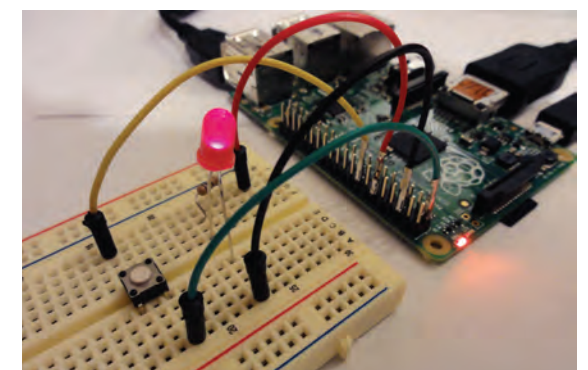


図5



## 入出力機器へのアクセス

## Raspberry Pi で

## BlueJ を使用すれば、

デスクトップでの作業と同じように標準的な Java プログラムを記述できるばかりか、Raspberry Pi の入出力ポートにもアクセスできます。

## プログラムの記述

前項までに、BlueJ の対話型機能を使用して、コードを1行も記述せずに入出力を簡単にテストすることができました。次に、GPOutput クラスと Button クラスのオブジェクトの利用方法について、単純なプログラムで確認していきます。ここでは、Raspberry Pi をライトのスイッチへと変えるコードを記述し、ボタンを押す間、LED が点灯するようにします。

す。

BlueJ プロジェクトで、LightSwitch という新しいクラスを作成します。最初に、LED 用のフィールドとボタン用のフィールドの2つを宣言します。

```
private GPOutput led;
private Button button;
```

次に、新しいクラスのコンストラクタ内で、これら2つの型のオブジェクトを作成し、さらに LightSwitch オブジェクト自体をボタンのリスナーとして登録します。

```
public LightSwitch()
{
    led = new GPOutput();
    button = new Button();
    button.addListener(this);
}
```

このプログラムが正常に動作するためには、この LightSwitch クラスが ButtonListener インタフェースを適切に実装する必要があります。そのために、クラスのヘッダー部で以下のように宣言します。

```
public class LightSwitch
    implements ButtonListener
```

また、buttonEvent メソッドも実装する必要があります。

```
@Override
public void buttonEvent(
    Button button)
{
    // code to come here
}
```

このリスナー・メソッドは、ボタンの状態が変わるたびに呼び出されます。ボタンの状態が変わるのは、ボタンを押したとき、およびボタンを離れたときです。button パラメータを使用することにより、ボタンの状態を確認できます。

残る作業は、ボタンが押されたときに、ライトの点灯または消灯という正しい応答を行うことです。そのために、buttonEvent メソッドの内部に以下のコードを記述します。

## リスト1

```
public class LightSwitch implements ButtonListener
{
    private GPOutput led;
    private Button button;

    /**
     * Create a Raspberry Pi light switch.
     */
    public LightSwitch()
    {
        led = new GPOutput();
        button = new Button();
        button.addListener(this);
    }

    /**
     * Listener method: called when button state changes.
     */
    public void buttonEvent(Button button)
    {
        if (button.isDown()) {
            led.on();
        }
        else {
            led.off();
        }
    }
}
```



すべてのリストのテキストをダウンロード





```

if (button.isDown()) {
    led.on();
}
else {
    led.off();
}

```

動作を確認しましょう。クラスを完成させ、コンパイルして、[LightSwitch](#) 型のオブジェクトを作成します（問題が発生した場合は、本記事で説明したクラスの完全な記述が[リスト 1](#)にありますので、自分のコードと比較してください）。このオブジェクトはリスナーとして登録されているため、メソッドを直接呼び出す必要はありません。ボタンを押すだけで、LED が点灯します。本当に、それほど簡単なのです。

### まとめ

Raspberry Pi で BlueJ を使用すれば、デスクトップでの作業と同じように標準的な Java プログラムを記述できるばかりか、Raspberry Pi の入出力ポートにもアクセスできます。本記事で見てきた例では、出力ピンに接続された LED を表すラッパー・クラスと、入力ピンに接続されたボタンを表すラッパー・クラスの 2 つを BlueJ で使用しました。これらのラッパー・クラスによって、対応する部品の操作、そしてコードの記述が非常に簡単になりました。

BlueJ は Raspberry Pi と通信するために、Pi4J ライブラリを内部的に使用してします。BlueJ のラッパー・クラスを使用してハードウェアと通信するほか、Pi4J インタフェースを直接使用し

てプログラムすることもできます (Pi4J へのアクセス方法の例については、[GPOutput](#) クラスのコードを参照してください)。

一般に、BlueJ のラッパー・クラスを使用するほうが、ハードウェアへの簡易なインタフェースが利用できるのが簡単です。ほかにもさまざまなラッパー・クラスが [BlueJ の Web サイト](#) で紹介されています。ただし、簡易性には柔軟性の犠牲がつきものです。BlueJ に提供されるラッパー・クラスよりも精細にハードウェアを制御する必要がある場合は、Pi4J ライブラリに直接アクセスしてください。このように、BlueJ はユーザーのレベルに応じて使用できます。学習を始めたばかりの人が簡単に足を踏み入れることができ、やや複雑なサンプルのプログラミングに慣れた後は柔軟に開発できるようになっています。

ですから、レベルを問わず、Raspberry Pi での Java 開発をぜひ始めてみてください。</article>

### LEARN MORE

- [BlueJ](#)
- [BlueJ on the Raspberry Pi](#)
- [Pi4J](#)
- [Raspberry Pi](#)

# Learn Java 8 From the Source Oracle University

- ✓ New Java SE 8 training and certification
- ✓ Available online or in the classroom
- ✓ Taught by Oracle experts
- ✓ 100% student satisfaction program

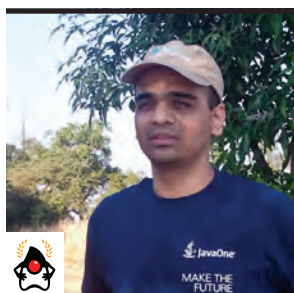


Learn More

ORACLE®







HARSHAD OAK

# Oracle Developer Cloud Service 入門

## 開発環境全体をクラウドで運用する

## Harshad Oak:

IndicThreads の設立者。Oracle JDeveloper や Jakarta Commons に関する著書あり。『Java 2 Enterprise Edition 1.4 Bible』(Wiley、2003 年) の共著者でもある。現在は『[Java EE Applications on the Oracle Java Cloud](#)』(Oracle Press) を執筆中。Java Champion、Oracle ACE Director であり、インドのプネーを拠点として活動している。



現在、Infrastructure as a Service (IaaS)、Platform as a Service (PaaS)、Software as a Service (SaaS) というクラウドのパラダイムは十分に定着し、ほとんどのクラウド・ベンダーはこれらのサービスの枠を超えて、ソフトウェア構築 / 運用における他のさまざまな面に対応しようとしています。ソフトウェア開発に必要なツールやテクノロジーの大半が Development Environment as a Service (DEaaS) としてクラウドで利用できるようになるのも時間の問題です。

開発者はすでに何らかの形でクラウドを利用していますが、ほとんどの場合、実際の開発環境はいまだにクラウド以外の場

クラウドは今後も IDE はオンで実行しようとする。外のクラウドに運命を委ねる。

所に構築されています。プロジェクト管理からバージョン管理、テスト、バグ追跡、コード・レビュー、継続的インテグレーションに至るまで、開発プロセスは依然として自社で運用されています。さらに、複数のベンダーによる無数の製品を組み合わせて、1つのソフトウェア開発環境を構成することもあります。

クラウドでの開発  
今後開発者の  
IDE は各自のマシ  
ンで実行されるで  
しょうが、それ以  
外のすべてがクラ  
ウドに移行され  
る運命にあります。

これこそ、新たな DEaaS ソリューションがターゲットにしている状況です。DEaaS は、それ自体は画期的なアイデアではなく、おそらく読者の皆さんも何らかの形ですでに見ており、さまざまな呼称で言及しています。しかし、今までと異なる点は、開発者やテクノロジーがリモート

のクラウドベース環境を十分利用できる水準にあり、今の新しいサービスが開発環境の一部だけでなく全体のクラウド化を目指していることです。

今後も開発者の統合開発環境 (IDE) は各自のマシンで実行されるでしょうが、それ以外のすべてがクラウドに移行される運命にあります。IDE もいつかはクラウドに移行されるかもしれませんが、現時点では、クラウドベースの IDE は NetBeans、Eclipse、Oracle JDeveloper といった機能豊富なデスクトップ IDE と比較すると見劣りします。

クラウド・ソリューションによるコスト削減効果のほか、に注目すべき点は、クラウドベース開発環境によって、開発環境の運用や管理に明示的、暗黙的に関わるチーム内のメンバーの負担がなくなることです。開発環境に問題が発生した場合に、メンバーに

大声で伝える必要はもうありません。クラウド・ベンダーに電話すれば良いわけです。チームのすべての開発者がソリューションの構築に集中できるようになり、周辺サービスの運用やインフラストラクチャのプロビジョニングにエネルギーを費やす必要がなくなります。

ソリューションは開発のライフサイクル全体を支えるべき

そのようなクラウド開発環境にとって最大の課題のひとつは、開発者のコラボレーションを後押ししながら、ソフトウェア開発のライフサイクル全体を支える必要があることです。また、クラウド開発環境は、開発環境以外の手段でプロジェクト関連のドキュメントを共有したいという開発者のニーズにも対応する必要があります。



しかし、そうは言うものの、開発のライフサイクル全体の構成要素に関する定義自体が数年ごとに変わり続けています。そのため、肝心なのは、クラウドベースの開発環境が次々に登場する新しい開発技術に適応できることです。過去5年から10年の間に、ビルド、継続的インテグレーション、ソース管理の分野で幅広く普及したツールを思い浮かべてみてください。今も人気のツールもあれば、別のツールに置き換わったツールもあるのではないのでしょうか。

それではここから、オラクルのクラウドベースの開発環境である Oracle Developer Cloud Service の特徴について詳しく見ていきます。

# Oracle Developer Cloud Service

オラクルは近年、クラウドに多大な力を注いできました。多数ある SaaS 関連製品から Oracle Database、Oracle WebLogic Server に至るまで、現在のオラクル製品のほとんどはクラウドに対応しており、しかも多くの場合カスタマイズ可能な形で提供されています。しかし、Oracle Developer Cloud Service は、既存の製品がクラウドに移行されたものではなく、最新の開発環境のニーズに対応して構築された新製品であるという意味で、他製品とは一線を画しています。

新しいサービスには、下位互換性の問題がなく、また、ごく一部の

開発者のみが要求するような古いテクノロジーをサポートする必要もないという大きな利点があります。そのため、既存の製品から発展したクラウド・サービスとは異なり、Oracle Developer Cloud Service には、クリーンで、最小限主義のような感覚があります。雑多な機能は最小限に抑え、2015 年の開発環境に期待される機能のみをサポートしています。

Oracle Developer Cloud Service  
の主な機能は以下のとおりです。

- GIT を使用したソース管理  
GitHub リポジトリとの統合
  - Maven リポジトリのサポート
  - タスク / 問題管理 (バグやタスクとコードのリンク機能付き)
  - Hudson を使用した継続的ビルド・インテグレーション
  - Oracle Java Cloud Service への直接デプロイ、またはオンプレミスでのデプロイ
  - Eclipse、NetBeans、Oracle JDeveloper との IDE 統合
  - ユーザー / チーム管理機能
  - Wiki ベースのコラボレーション機能
  - コラボレーション機能とフィルタ機能を搭載したコード・レビュー
- 現在、Oracle

Developer Cloud Serviceでは、一般的なタスクやフローに対応した数種類のテンプレートが提供されています。しかし、クラウド開発環境では何百ものプロジェクトにわたって利用状況が分析されるため、今後の Oracle Developer Cloud Serviceでは、ソフトウェア開発のライフサイクルにおける多数の定型的作業に対応したテンプレートやベスト・プラクティスが提供されることを期待したいと思います。

Oracle Developer Cloud Service  
自体に画期的なところは何もあり  
ませんが、開発環境全体をクラウ

ドで運用できるようになったというところが画期的です。また、新しいプロジェクトを作成する際に、Oracle Developer Cloud Service スタック全体がプロビジョニングされます。開発環境を自分で構築する時間と比較すると、ほんのわずかな時間しかかかりません。

## 試用版の利用

Oracle Developer Cloud Serviceは現在、Oracle Java Cloud Service - SaaS Extension の試用版と有償版、および Oracle Messaging Cloud Service の有償版に無償で

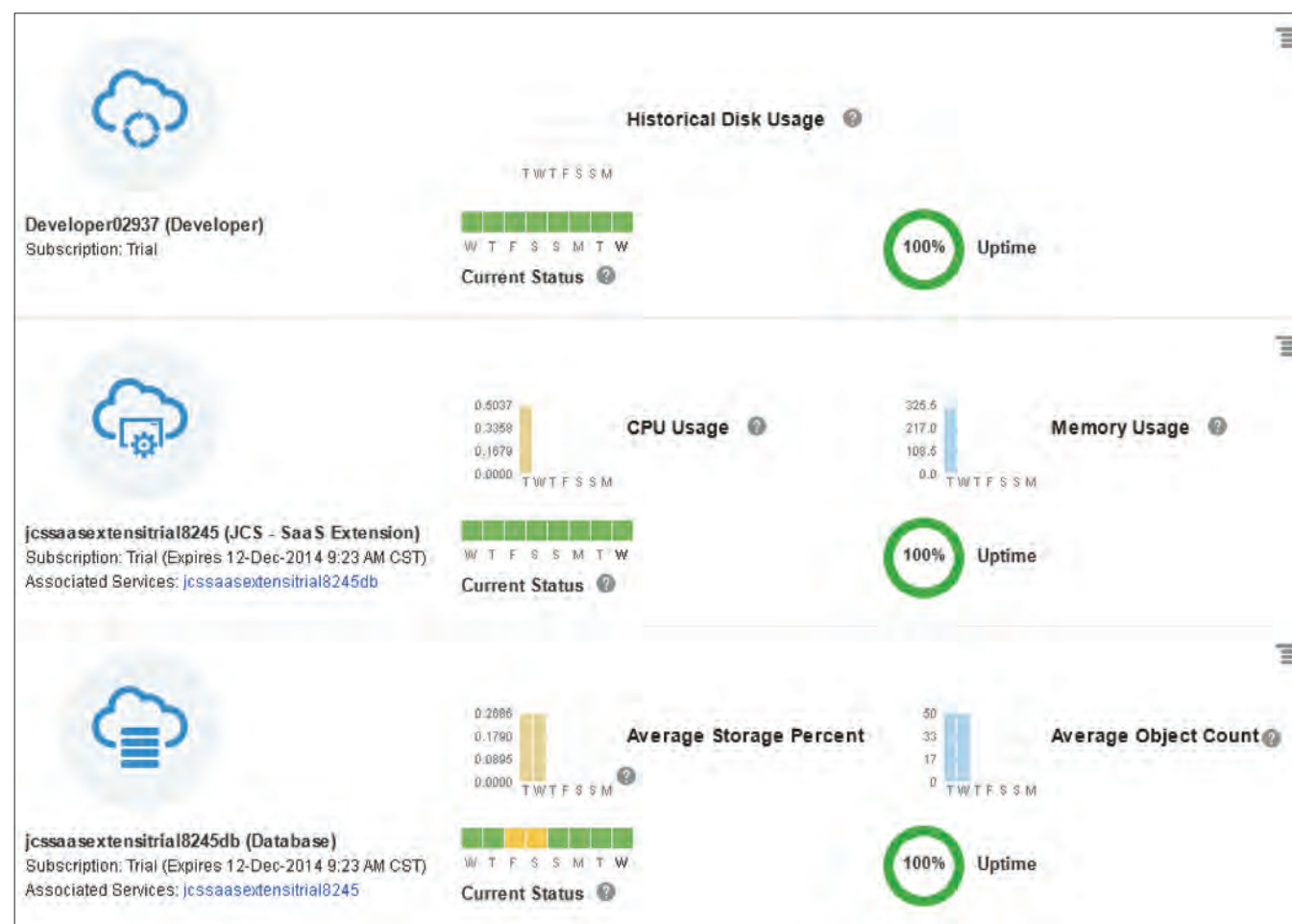


图1





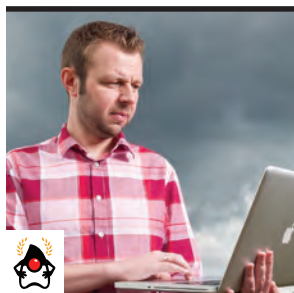












BERT ERTMAN

**Bert Ertman**

(@BertErtman、bert.ertman@luminis.eu) : Luminis のフェロー、Java Champion。『Building Modular Cloud Apps with OSGi』の共著者。Dutch Java User Group (NLJUG) の代表者でもあり、Java およびソフトウェア・アーキテクチャに関して多数の講演を行っている。JavaOne 2012 Rock Star、2013 年の Duke's Choice Award を受賞。



写真: TON HENDRIKS

# モジュール式クラウド・アプリケーションを Java で構築する

アーキテクチャに手を加えず保守性を維持しながら、モジュールを追加、置換、削除

ここ数年にわたり、勤め先である Luminis の同僚と共に、OSGi ベースの開発スタックを用いるダイナミックなサービス・アーキテクチャに合わせたクラウド・アプリケーションを構築してきました。OSGi と聞いて、一体なぜそんなものが必要なのかと思う方もいるでしょう。クラウドはモジュール式アプリケーションと非常に相性が良いということが分かっています。そのモジュール式アプリケーションに必要なのが OSGi なのです。

Luminis の一部の顧客はクラウドを採用しています。それは、新しい市場や変わりつつある市場での「先発者」となることを望んでいるからです。そのほかにも、インフラストラクチャに対して「利用した分だけ支払う従量課金」を利用したいと思っています。そのような顧客向けに構築するアプリケーションの要件は、速いペースで変更される傾向にあります。

コードベースの変更に対処す

ることは、簡単ではありません。そのため、筆者らは最初から完璧なシステムを設計するのではなく(そのような設計は不可能です)、アーキテクチャを小さく、使い捨て可能な、相互に関連するモジュールへと分割することにしました。そうすれば、アーキテクチャに手を加えず保守性を維持しながら、アプリケーションの一部分を自在に追加、置換、削除できるようになります。また、そのようなモジュール化によって、アーキテクチャのリファクタリングの際に、アプリケーションに影響を与えずに変更に対処するメカニズムが確立します。

**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[こちら](#)からダウンロードできます。

**モジュール化の背景**

モジュール化は新しい考え方ではなく、かなり昔から存在していました。modularity (モジュール化) とい

う用語が作り出されたのは 1960 年代後半のことで、Edsger W. Dijkstra 教授が論文の草稿において初めて言及しました。

残念ながら、Java プラットフォーム自体に、モジュール・システムは備わっていません。現在、Project Jigsaw が進行中ですが、本番用途にはまだ対応していません。私たちはモジュール式ソリューションをすぐに必要としていたため、OSGi を選択しました。OSGi には、動的なモジュール式ランタイムを Java 仮想マシン (JVM) の上で実行できるという利点があります。OSGi は、10 年を超える実績のあるモジュール式ソリューションです。

**結合を避け、凝集を促す**

モジュール化とは、(密に) 結合されたクラスを分離するための手法です。モジュール (OSGi の用語ではバンドル) 内にクラスを配置することにより、クラスを分離できます。そして、他のモジュールからインポートするイン

タフェースやクラスに対して厳密な依存性を定義できます(凝集度)。

これらのモジュールの内部でサービスを定義し、一元化された 1 つのサービス・レジストリにそれらのサービスを登録します。ユーザーはこのサービス・レジストリ経由でサービスへの参照を取得します。実際には、このメカニズムは Spring、Contexts and Dependency Injection (CDI) などの依存性注入フレームワークに少し似ています。

**図 1** は、モジュールが他のモジュールに依存している様子を示しています。共有クラスと非公開クラスが明示されています。**図 2** は、サービス・リポジトリによる制御の反転の実装方法を示しています。

**API の定義**

モジュール式インフラストラクチャの設計時に、通常は API モジュールを定義することから始めます。システム内の他のモジュール











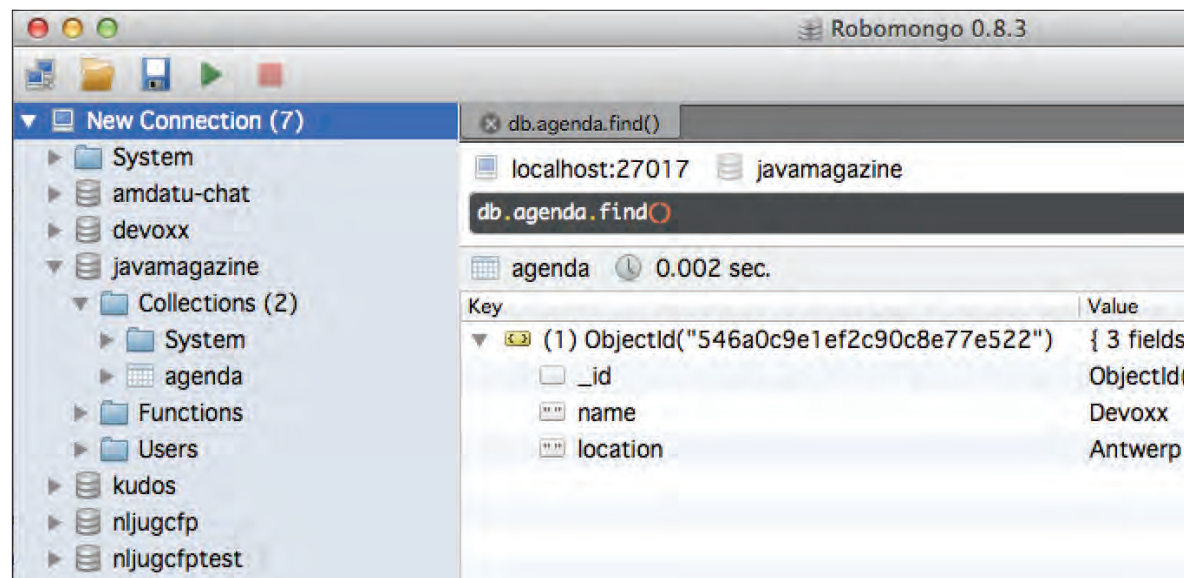


図4

つの実装を選択するためのさまざまな機能を提供しています。たとえば、サービスの優先順位に基づいた選択や、サービス・レジストリに受け渡しできるフィルタ条件に基づいた選択が可能です。

別のバージョンのサービスを利用可能な状態にしておけば、プライマリ・サービスがダウンした場合に、多少の品質低下はあり得るものの、運用を続行することもできます。OSGiは、保守作業や新しいバージョンへのアップグレードなどでプライマリ・サービスが（一時的に）利用できなくなった場合に、サービスの実装を自動的に切り替えます。この機能が必要ない場合は、単に古い実装を捨てるという選択肢もあります。システム内の他の要素はAPIにのみ依存しているため、他の要素に影響を与えずに古い実装を捨てることができます。この点が、このモジュール式の開発アプローチを利用する上で使い捨て可能なコンポーネントがいかに機能する

かを示す一例です。

### プロビジョニングとデプロイ

デプロイ時に1つのWARファイルだけでなく、多種多様なモジュール（JARファイル）とそれらの依存性を扱うケースにどう対処するのだろうと思った方もいるでしょう。

Luminisの一部の本番システムでは、300近くのJARファイルをデプロイする必要があります。これらのJARファイルと各ファイルのバージョンを記録するために、いわゆるプロビジョニング・システムとしてApache ACEを利用しています。

ACEは、モジュールを記録し、セマンティックなバージョン情報を認識できる巨大リポジトリであると考えられます。ACEでは、機能セットとディストリビューションを定義できます。機能セットには、バージョン管理された多数のモジュールが含まれ、ディストリビューションには複数の機能セットが含

## リスト7 リスト8

```
package agenda.mongo;
```

```
public class MongoAgendaService implements Agenda {
    private volatile MongoDBService mongoDBService;
    private volatile Jongo jongo;
    private volatile MongoCollection agenda;
```

```
    public void start() {
        jongo = new Jongo(mongoDBService.getDB());
        agenda = jongo.getCollection("agenda");
    }
```

```
    @Override
    public void addConference(String name, String location) {
        agenda.save(new Conference(name, location));
    }
```

```
    @Override
    public List<Conference> listConferences() {
        List<Conference> result = new ArrayList<>();
        agenda.find().as(Conference.class).forEach(result::add);
        return result;
    }
}
```

 [すべてのリストのテキストをダウンロード](#)

まれます。

クラウド環境においては、管理エージェントのみを利用してACEに接続するためのノードと、ソフトウェア・ディストリビューションがオンラインになった際にそれをダウンロードするためのノードを構成します。アプリケーションをモジュール式にすることの利点は、後で1つのモジュールだけを変更し、そのモジュールをACEにアップロードするとす

ぐに、ACEによってその差分のみ（つまり、変更されたモジュールのみ）が運用中のノードに発行されることです。

私たちはこのようなプロビジョニング・アプローチを、クラウドだけでなく、デバイスに対しても利用しています。Felixを実行できるあらゆるデバイスが、このアプローチに適しています。図5に、プロビジョニング・サーバーとしてのApache ACEの例を示します。



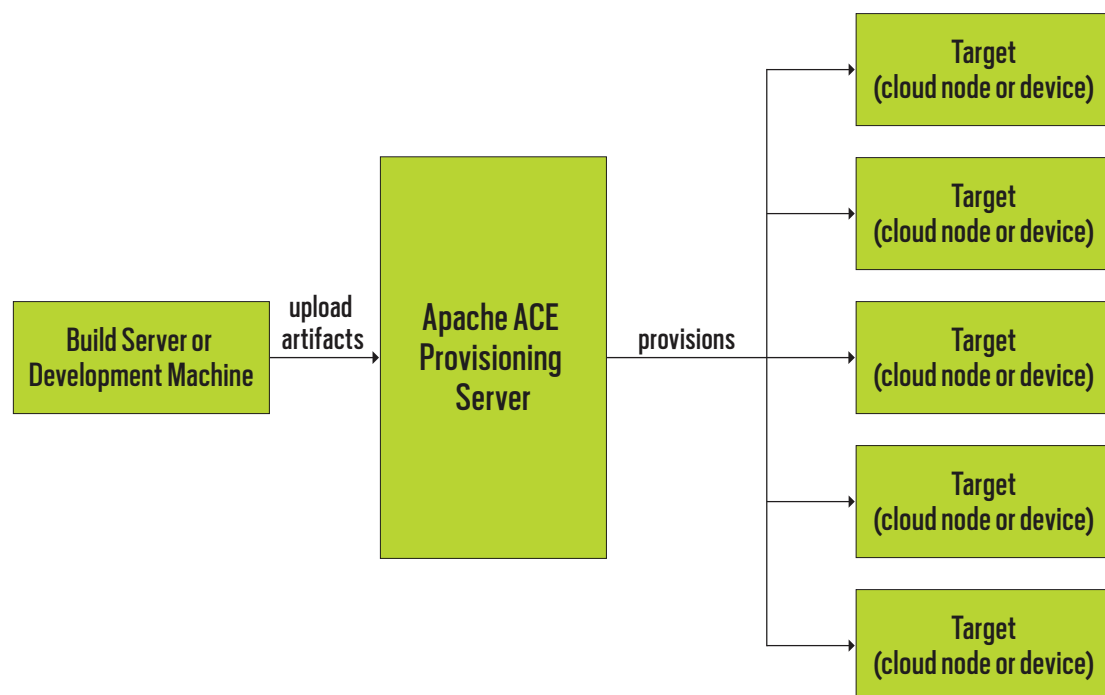


図5

### デプロイメントに関する追加情報

皆さんは Java EE 環境でのアプリケーション・サーバーの概念に詳しいはずですが。私たちは、Felix だけによる軽量で最小限の実装を導入しましたが、アプリケーションをアプリケーション・サーバーにデプロイすることも十分に可能です。今日のほとんどのアプリケーション・サーバーは、内部的にモジュール化に対応した OSGi ベースのアーキテクチャを確立しているからです。GlassFish や IBM WebSphere などのアプリケーション・サーバーでは、通常の WAR ファイルや EAR ファイルと同様に、OSGi バンドルをアプリケーション・サーバーにデプロイできます。

スケーリングのためのアーキテクチャクラウドはスケーリングを見越したアプリケーション・アーキテクチャの手法で

もあるため、各ノードをステートレスとして維持する必要があります。クラウド・インフラストラクチャ・プロバイダ（Amazon や Azure など）のロードバランサを利用すれば、クラスタの状態を監視できます。ノードに異常が発生した場合や、処理能力の拡張が必要になった場合に、ロードバランサによって新しいノードを起動できます。起動されたノードは自動的に ACE に接続して最新のディストリビューションをダウンロードし、クラスタに参加します。

この同じアプローチを利用して、待機状態のサーバーの費用を抑えるためにスケールダウンすることも可能です。このようにして、本当の意味で柔軟なスケーリング・アプローチを確立できます。

ステートレス・アーキテクチャを利用するということは、何らかのデータストアでそのデータを管理する必要がある

ことも意味します。実際、私たちはさまざまなソリューションを利用しています（この状況は多言語永続化と呼ばれます）。たとえば、ドキュメントを中心としたやり取りには MongoDB を利用することが多いですが、リレーショナルストレージが必要な場合は、リレーショナルなバックエンド・ストアとして JPA ベースのアプローチを採用します。大きいバイナリ・データ、スキャン、画像などに対しては、BLOB ストアを利用します。Amdatu は、これらの多種多様なストアに接続するためのコンポーネントを提供しています。

### マイクロサービスとの比較

最後に、本記事で紹介したモジュール式アプローチと、最近話題になっているマイクロサービス・アーキテクチャとの違いについて触れておきます。サンプル・コードからわかるように、この実装の最終的な結果は、（コード行数の観点で言えば）非常に簡潔です。この最終的な結果は、スタンドアロン・デプロイメントとして公開され、1つのプロセス内で実行され、1つの処理を完遂します。

言うまでもなく、サービスのレジリエンスの確保や監視など、対処すべき非機能要件はほかにもありますが、ライブラリをいくつか追加すれば、これらのニーズの大部分に対応できます。言い換えれば、本記事で説明したようなモジュール式開発アプローチは、マイクロサービスベースのアーキテクチャとよく適合しますが、それに加えて、アーキテクチャ全体をダウンさせることなく実装のごく一部だけを更新できるという特

徴があります。

### まとめ

本記事では、Java でクラウド・アプリケーションを扱う方法、そして特にモジュールとバージョンを認識するプロビジョニング・システムと併用したときにモジュール式開発アプローチがクラウドと非常に相性が良い理由について理解していただけたと思います。

一般に、モジュール化は究極のアジャイル・ツールであると考えます。モジュール化は、変化し続けるコードベースへの対処に有効です。また、すべてのサービス実装が使い捨て可能なコンポーネントとなり、アプリケーションの他の部分に影響を与えずに追加、置換、削除できるため、長期的な保守に関する心配事を減らすことができます。</article>

MORE ON TOPIC:



### LEARN MORE

- [Building Modular Java Applications in the Cloud Age \(ビデオ\)](#)
- [Building Modular Cloud Apps with OSGi](#)
- [モジュール式アプリケーションの構築に関する JavaOne 2013 プレゼンテーション](#)
- [その他のサンプル・アプリケーション](#)







参照)。

## Java EE 7 によるコード削減

EJB ベースの `Executor` のカスタム実装には、大きな欠点があります。並列リクエスト数の制限や、基盤となるスレッド・プールの設定変更を行う標準的な方法がないことです。JSR 236 (Concurrency Utilities for Java EE) に対応した Java EE 7 には、Java SE の `java.util.concurrent.ExecutorService` のマネージド実装が付属しています。`ExecutorService` は `Executor` インタフェースを拡張しており、その結果、カスタム EJB 実装は過去のものとなりました。

### リスト 6 の

`ManagedExecutorService` が、`ExecutorService` のマネージド実装です。これはアプリケーション・サーバーのマネージド・リソースであり、`@Resource` アノテーションによってインジェクションされます。

**リスト 7** は、POJO の内部で `String` を加工する例で、保存の前に `Normalizer#normalize` メソッドで入力を処理しています。

Java SE 8 では、`java.util.function` パッケージに便利な SAM が数多く組み込まれています。**リスト 8** に示すように、`Consumer<String>`

## コードの削減

Java SE 8 の力を少し借りると、複数のコンポーネントを連結して、コードを 1 行に減らすことができます。

同時に、Java EE 7 が実用的な機能を提供しています。

インタフェースは `Storage#store` メソッドの抽象化に使用でき、`Function<String,String>` は `Normalizer#normalize` メソッドのプレースホルダーとして使用できます。

### Java SE 8 でさらにコードを削減

Java SE 8 には、協調動作をサポートする魅力的なユーティリティとして `java.util.concurrent.CompletableFuture` クラスが付属しています。`CompletableFuture` は `Supplier` インスタンスを `Function` や `Consumer`

に接続し、柔軟な実行パイプラインを実現します。**リスト 8** の例は、`CompletableFuture` によって大幅に簡略化できます。**リスト 9** では、`Storage` と `Normalizer` を `CompletableFuture` で接続しています。

`Function` インタフェースと `Consumer` インタフェースへの割り当ては、単なるデモの目的で行われています。なお、**リスト 9** のコードは、アプリケーション・サーバーのマネージド・スレッド・プールでは実行できません。このコードを実行できるのは、「共有プール」と呼ばれる `ForkJoinPool` のインスタンス `ForkJoinPool.commonPool()` です。名前が「Async」で終わるすべ

リスト1

リスト2

リスト3

リスト4

リスト5

リスト6

```
@Stateless
public class StorageService {
    @Asynchronous
    public void save(String content) {
        //heavy lifting
        System.out.println("Storing " + content);
    }
}
```

すべてのリストのテキストをダウンロード

でのメソッドには、2 つ目のパラメータとして `Executor` インスタンスを渡せるようになっています。`ManagedExecutorService` は `Executor` であり、マネージド・アプリケーション・サーバーのスレッドでパイプラインを実行するために使用できます。**リスト 10** では、`Executor` として `ManagedExecutorService` を渡し、SAM としてメソッド・ハンドルを使用しています。

Java SE 8 の力を少し借りると、複数のコンポーネントを連結して、コードを 1 行に減らすことができます。それと同時に、Java EE 7 が、`ManagedExecutorService` の管理、監視、設定などの実用的な機能を提供しています。

### 発生する例外

**リスト 10** のパイプラインで発生し

た例外はすべて、通知されることなく飲み込まれてしまいます。この「便りが無いのは悪い知らせ」的な振る舞いは、出発点としてはよくても、大半のユースケースでは許容できないでしょう。外側の `StorageService#save` メソッドは、パイプラインがタスクを処理する前に終了する可能性があります。`CompletableFuture` は、ブロッキング式の例外処理と、非同期式の例外処理の両方を提供しています。`Future#get` メソッドは呼出し元をブロックし、戻り値を返すか、または `ExecutionException` を返して発生したエラー原因を知らせます。**リスト 11** は、例外のブロッキングの例です。

**リスト 11** の `get()` メソッドによって例外は飲み込まれなくなりますが、呼び出すメソッドは `CompletableFuture` がすべてのタス



クを終えるまでブロックされてしまいます。**リスト 12** は、非同期式の例外処理の例です。例外発生時に非同期で呼び出される `exceptionally()` メソッドには、パラメータとして `Function<Throwable,[RETURN_VALUE]>` を渡します。

`exceptionally` メソッドは、`RETURN_VALUE` の型が様々であり、任意の段階で適用できます。たとえば、`supplyAsync` や `thenApplyAsync` の直後では、2 つ目の総称型パラメータは `String` となります。このメソッドに渡すのは SAM であり、メソッド・ハンドラによって簡単に実装できます。エラー処理コードを専用のハンドラに移すことによって、さらにコードを効率化できます。**リスト 13** では、専用ハンドラで例外処理を行っています。

例外は非同期的に発生するため、`StorageService` クラス内で意味のある処理を行うことはできません。そのため、例外処理は完全に専用ハンドラにまとめることができます。`save` メソッドの実装は、処理を呼び出すだけで結果は考慮しない形になっています。エラー処理戦略は、メッセージ駆動型 Bean (MDB) や Java Message Service (JMS) と同様です。

非同期コードのエラー

を呼び出し元に伝えることはできません。そのため、エラーはデッド・レター・キューとも呼ばれる専用の JMS キューに送られます。`CompletableFuture` にも、同じメカニズムを JMS を使用せずに適用できます。非同期の例外処理には、単純な Contexts and Dependency Injection (CDI) イベントがもっとも適しています。**リスト 14** では、例外をイベントとして送信しています。

補足すると、`javax.enterprise.event.Event#fire` メソッドは、総称型の `Consumer<Object>` です。このメソッドを使用すると、`Storage` への直接参照を、インジェクションされた別の `Event` インスタンスで完全に置き換えることができます。**リス**

**ト 15** では、CDI イベントを `Consumer<String>` として使用しています。

`Storage` クラスは、ほとんどそのまま使用できます。`save` メソッドの `String` パラメータに `@Observes` アノテーションを追加する必要があるだけです。**リスト 16** では、`Storage` クラスを CDI リスナーにしています。


### 非同期 JAX-RS 2.0

現在のところ、JAX-RS リソースでは大したことはしていません。**リスト 17** の `@POST` メソッドと `@GET` メソッドは、いずれも同期的に実行されま

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 class Normalizer {
    public String normalize(String input) {
        return "#" + input;
    }
}
//...
@Inject
Normalizer normalizer;

public void save(String input) {
    String content = normalizer.normalize(input);
    executor.execute(() -> storage.store(content));
}
```

 すべてのリストのテキストをダウンロード

す。

JAX-RS 2.0 では、非同期リクエスト処理のために `AsyncResponse` パラメータが導入されました。`AsyncResponse` に `@Suspended` アノテーションを追加すると、リソースのメソッドはリクエストが完了する前に終了できるようになります。このアプローチによって、実際のビジネス・ロジックの処理から接続処理スレッドを分離できるようになります。**リスト 18** は、そのような非同期リクエスト処理の例です。

待機中のリクエストは、`AsyncResponse#resume` メソッドの呼び出しによって再開できます。通信プロトコルはビジネス・ロジックの実装から分離した方が良いため、ビジネス・ロジックに `AsyncResponse` を公開するべきではありません。そ

のため、ビジネス・ロジックはこの外側で実装します。ビジネス・ロジックに `AsyncResponse` を送信する際は、その存在を隠ぺいしたまま送信する必要があります。

Java SE 8 より前では、`AsyncResponse` をカスタム・ラッパー内でカプセル化することが一般的でした。しかし、Java SE 8 では、`AsyncResponse#resume` メソッドを `Consumer<String>` として送信するだけで大丈夫です。`AsyncResponse` をビジネス・ロジックと密接に統合する際に、抽象化やインタフェースの追加を行う必要ありません。

また、`@POST` メソッドを手直して、`StorageService` への参照を完全に取り除くこともできます。`StorageResource#store` メソッ

### 2 つのパワー

Java SE 8 がアプリケーション・アーキテクチャに与えるインパクトと比べれば、Java EE 7 によるインパクトはわずかなものです。本当に面白くなるのは、この 2 つを組み合わせたときです。Java SE 8 のパワーが Java EE 7 の生産性と融合します。





ドは入力パラメータをバックエンドに送信します。バックエンドは、[Supplier<String>](#) によって簡単に抽象化できます。パラメータはラムダ式を使用して即座に [Supplier](#) に変換され、通常の CDI イベントとして送信されます。すでに JAX-RS リソースに存在している Java SE 8 ネイティブのインタフェースを使用すると、[StorageService](#) のコードをさらに簡略化できます。**リスト 19** では、非同期に [String](#) を配信しています。

**リスト 19** の [getContent](#) メソッドは、[Storage#retrieve](#) メソッドを [Supplier<String>](#) として使用し、その出力を [Consumer](#)

プタは、個々のタスク（制御）や境界のモニタリングに非常に適しています。**リスト 20** は単純なパフォーマンス測定インターセプタです。

アノテーションは 1 つだけ ([@Interceptors\(Monitor.class\)](#)) でよく、アノテーションが付加されたクラスのすべてのパブリック・メソッドがインターセプトされてモニタリングされます。**リスト 20** の [Monitor](#) インターセプタは、ロギングに [System.out.println](#) を使用しています。実環境では、代わりに CDI イベントを使用してモニタリング・データをセンターに送信し、今後の分析に役立てることができます。

### Java SE 8 による JPA コードの削除

Java Persistence API (JPA) を使用すると、[Store](#) クラスを拡張して簡単に永続化機能を持たせることができます。**リスト 21** では、簡単な JPA 2.1 コードによって永続化を行っています。各ユーザーに対して、[String](#) の [content](#) がそれぞれ格納されています。

ここで統計をとってみます。例として、ユーザー 1 人当たりの平均 [String](#) サイズを計算します。このような計算は、CPU に負荷がかかる可能性があるため、並列処理を行います。JPA 問合せには、ストリーム可能な Java SE 8 コレクションが返されます。Java SE 8 コレクションによって、インメモリの処理や分析を Java SE 8 でシンプルに行う道が開かれます。**リスト 22** では、単純な [Store](#) の統計を作成しています。

[Object](#) の入力として接続します。また、この例では、[CompletableFuture](#) とインジェクションされた [ManagedExecutorService](#) をパイプラインとして使用しています。

### Java EE 7 によるモニタリング

非同期アプリケーションでは、同期アプリケーションよりもモニタリングやデバッグが難しくなります。Java EE 5 で導入されたインターセ

### モニタリング

非同期アプリケーションでは、同期アプリケーションよりもモニタリングやデバッグが難しくなります。インターセプタは、個々のタスクや境界のモニタリングに非常に適しています。

リスト13

リスト14

リスト15


リスト16

リスト17

リスト18

```
public void save(String input) {
    CompletableFuture.supplyAsync(() -> input, executor)
        .thenApplyAsync(normalizer::normalize, executor)
        .thenAcceptAsync(storage::store, executor)
        .exceptionally(this::handle);
}

public Void handle(Throwable error) {
    System.out.println("error = " + error);
    return null;
}
```

 [すべてのリストのテキストをダウンロード](#)

結果リスト全体が RAM（日々安くなっています）に収まるなら、**リスト 22** のように、JPA エンティティ上で Java SE 8 のラムダ式を使用し、簡単に実装やテスト、問合せの実行を行うことができます。Java SE 8 には、さまざまな [Collector](#) 統計ユーティリティが導入されており、コードを追加することなく必要な計算を行うことができます。[IntSummaryStatistics](#) ユーティリティ・クラスは、[JsonObject](#) として簡単に公開できます。たとえば、**リスト 23** では、

[IntSummaryStatistics](#) を [JsonObject](#) に変換しています。

並列ストリームにはスレッド管理のオーバーヘッドがかかるため、単純すぎるユースケースでは十分なパフォーマンスは得られません。また、多くのユースケースでは、**リスト 22** は単一スレッドの [stream](#) を使用する方が [parallelStream](#) を使用するよりパフォーマンスがおそらく良いでしょう。さらに、非同期 [parallelStream](#) は、アプリケーション・サーバーのマネージド・スレッドではなく、[ForkJoinPool](#) が提供す



るスレッドで実行されます。使用可能なフォーク/ジョイン・スレッドの数は使用可能な CPU の数によって異なりますが、システム・プロパティ [java.util.concurrent.ForkJoinPool.common.parallelism](http://java.util.concurrent.ForkJoinPool.common.parallelism) で設定できます。アプリケーション・サーバーは、このような「勝手に」作られたフォーク/ジョイン・スレッドは認識しないため、大量のスレッドによってパフォーマンスのボトルネックや堅牢性の問題が発生する可能性があります。

### まとめ

Java EE 7 は多くの注目を集めています。「Java EE」という名前がついているだけで、会議室は一杯になり、ブログ記事は人気になります。しかし、Java SE 8 がアプリケーション・アーキテクチャに与えるインパクトと比べれば、Java EE 7 によるインパクトはわずかなものです。本当に面白くなるのは、この2つを組み合わせたときです。Java SE 8 のパワーが Java EE 7 の生産性と融合します。

本記事の目的は、無駄なプログラミングをなくし、きれいなコードを書くためのアイデアを提供することです。また、本記事に掲載したサンプル・コードは大幅に簡略化しています。[String](#) の永続化も、Java SE 8 と Java EE 7 の API を最大限

に使用して行いました。しかし実際環境では、実際のユーザーには意味のないクールなテクノロジーを追い求めるよりも、中核となるビジネス・ロジックに集中するべきでしょう。

</article>

### 大変な注目 Java EE 7 は多くの 注目を集めています。

「Java EE」という名前がついているだけで、会議室は一杯になり、ブログ記事は人気になります。

### LEARN MORE

- [この記事で使用されたサンプル・プロジェクト](#)
- [JSR 346 : Contexts and Dependency Injection for Java EE 1.1](#)
- [「Going Asynchronous With CompletableFuture and Java EE 7」スクリーンキャスト](#)

リスト19

リスト20

リスト21

リスト22

リスト23

```
@Stateless
public class StorageService {
    @Resource
    ManagedExecutorService executor;
    @Inject
    Normalizer normalizer;
    @Inject
    Event<Throwable> exceptionHandler;
    @Inject
    Event<String> sink;
    @Inject
    Storage storage;

    public void save(@Observes Supplier<String> supplier) {
        CompletableFuture.supplyAsync(supplier, executor)
            .thenApplyAsync(normalizer::normalize, executor)
            .thenAcceptAsync(this.sink::fire, executor)
            .exceptionally(this::handle);
    }

    public Void handle(Throwable error) {
        exceptionHandler.fire(error);
        return null;
    }

    public void getContent(
        @Observes Consumer<Object> listener) {
        CompletableFuture.supplyAsync(
            thenAcceptAsync(listener, executor)
            .exceptionally(this::handle);
        }
    }
}
```



すべてのリストのテキストをダウンロード







JOHAN VOS

**Johan Vos:**  
1995 年から  
Java に関わる。  
ソーシャル・ネッ  
トワーキング・  
ソフトウェア向  
けの Java ベー  
ス・ソリューションに取り組む  
LodgON の共  
同創設者。組  
込み開発とエ  
ンタープライズ  
開発の両方に  
熱心であり、徹  
底して JavaFX  
と Java EE を使  
用したエンド・  
ツー・エンドの  
Java に専念。



写真: TON HENDRIKS

# クラウドベースの IoT デバイス・モニタリング

高い性能とスケーラビリティを両立したクラウドベースのモニタリング・システムで組み込みデバイスからデータを収集する

Internet of Things (IoT)、つまりインターネットに接続されている組み込みデバイスの数は日々増え続けています。こういったデバイスの多くは、さまざまなバックエンド・サービスとの間でデータを送受信するためにインターネット接続を必要とします。しかしそれ以外にも、インターネット接続は、デバイスやデバイスが実行しているソフトウェアを管理するためにも必要となる場合が多くあります。

デバイスを管理するためには、少なくともモニタリングができればなりません。オペレータ、つまりデバイスのメンテナンス責任者は、デバイスの現在の状態や履歴、現在の稼働状況などを知りたいと考えています。インターネットに接続されたデバイスの数が増える中で、そういったデバイスのモニタリングは大きく重要な課題になっています。本記事では、インターネット接続さ

れたデバイスをクラウドベースでモニタリングすることによって多数のキオスク・システムの運用を実現しているユースケースを詳しくみていきます。

## ユースケース

CultuurNet Vlaanderen は、ベルギーの政府組織です。フランドル地方の当局の要請によって、この組織は人々の文化に対する意識を高めようとしています。2012 年に、CultuurNet は UiTPAS プロジェクトを立ち上げました。UiTPAS は、近距離無線通信 (NFC) カードを用いたシステムで、ユーザーは文化行事に参加してポイントを貯め、そのポイントを景品と交換することができます。ユーザーは、カードをキオスク・システム(図1参照)でスキャンします。このシステムは、NFC リーダーと LCD スクリーンに接続された Raspberry Pi をベースに構築されています。

キオスク・システムは、チェックイン装置 (CID) とも呼ばれています。当初の目的はユーザーが文化活動に「チェックイン」することだったからです。当初、UiTPAS は 3 都市だけで展開され、約 30 台の CID が、劇場、映画館、博物館などの文化スポット周辺に設置されました。CID の運用は、3 都市に代わって CultuurNet が行いました。

3 都市での UiTPAS の成功が引き金となり、多くの都市が関心を示すようになりました。そのため、さらに多くの CID が製造され、運用スポットも増えました。現在、各都市は、自分たちでデバイスを管理・運用することを求められています。

明らかに、これ以上手作業で管理を続けるのは不可能です。また、分散管理の必要性(各都市が自分の所有しているデバイスのモニタリングや調整を行えることが必要)がある



図1

ことも、運用面の課題となっています。CultuurNet は、こういった課題に対処するため、登録されたすべての CID を監視し、さまざまなオペレータからのアクセスが可能なクラウドベースのモニタリング・システムを使用しています。各オペレータ(通常は各都市の職員である IT 管理者)が監視や管理を行えるのは、自分の市の CID のみです。

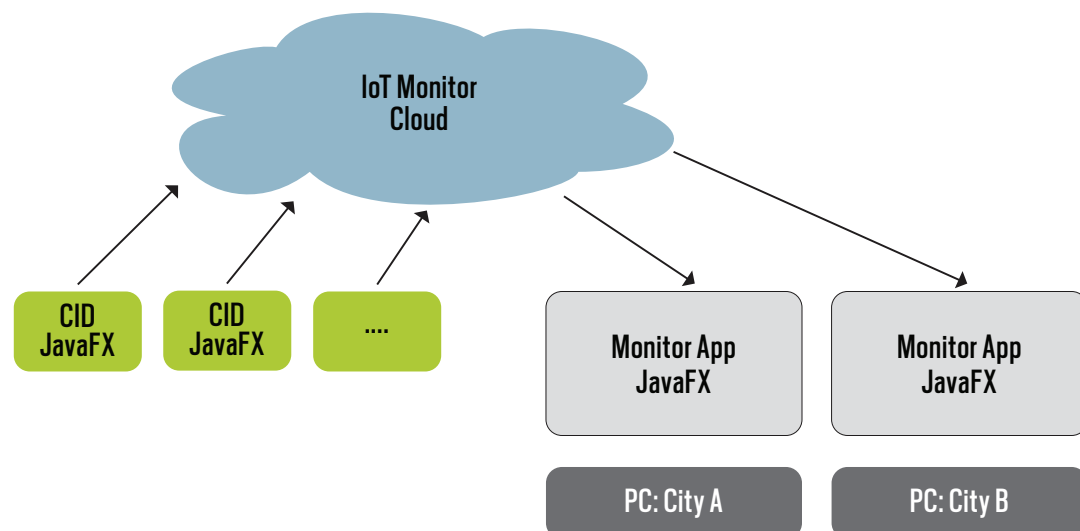


図2

このソリューションは、全面的に Java で構築されています。CID は、Raspberry Pi 上で JavaFX を実行し、IoT モニター・クラウドにモニタリング情報を送信します。IoT モニター・クラウドはこのアーキテクチャの中核であり、Java EE 7 (GlassFish) で実装されています。クライアントとなるモニタリング・アプリケーションも、JavaFX によるアプリケーションです。図2にシステム構成を示します。

各コンポーネントはすべて Java を使用していますが、それぞれの環境は異なります。IoT モニター・クラウドは、すべての CID からすべての情報を収集し、蓄積します。このコンポーネントは Java EE 7 API を使用しており、Amazon EC2 クラウドの GlassFish インスタンス内で実行されています。ストレージ機能として Amazon DynamoDB を、検索機能としてオープンソースの [Elasticsearch エンジン](#) を使用しています。

### CID からクラウドへの接続

CID は、動作イベントやロギング・データなどのモニタリング・データを IoT モニター・クラウドに送信します。ロギング・データは Java アプリケーションで簡単に取得できるため、この機能は任意のアプリケーションで使用できます。CID アプリケーションは、内部的に **リスト1** のロガーを使用しています。

すべてのシステム関連イベント（カードがスキャンされた、ネットワーク要求が実行された、ネットワーク要求がタイムアウトしたなど）は、たとえば次に示すような Java 標準のロギング・コマンドによってログに記録されます。

```
LOGGER.log(Level.SEVERE,
"Could not read card.");
```

**リスト2** に示すように、[LogHandler](#) を拡張した専用の [MonitorHandler](#) を作成して、ロガーに追加しています。この追加

リスト1 / リスト2 / リスト3 / リスト4

```
public static final Logger LOGGER =
Logger.getLogger("be.uitpas.pi");
```

すべてのリストのテキストをダウンロード

によって、[LOGGER](#) インスタンスで記録されるすべてのエントリが [monitorHandler](#) に通知されるようになります。

[MonitorHandler](#) は、**リスト3** に示すように定義されています。アプリケーションが [LOGGER.log\(..\)](#) を呼び出してログを記録すると、[MonitorHandler](#) の [publish](#) メソッドが呼び出され、ログ・メッセージにタイムスタンプ、スレッド情報、メソッドなどの追加情報が付加されます。

[MonitorHandler](#) は、DataFX を使用してログ情報を IoT モニター・クラウドに送信します。DataFX とは、JavaFX にエンタープライズ機能を持たせる JavaFX ベースのフレームワークです。DataFX コンポーネントの中

には、DataSources コンポーネントがあります。このコンポーネントがあることで、バックエンド・システムとの REST ベースの通信が容易になります。DataFX は、JavaFX のスレッドモデルに従ってバックグラウンド・スレッドで実行されます。アプリケーションへの折り返しには、JavaFX アプリケーション・スレッドを使用します。

**リスト4** のコードに、組み込みデバイスから IoT モニター・クラウドへ向けての REST リクエストを DataFX から実行する方法を示します。IoT モニター・クラウドのエンドポイントは、架空のアドレス <http://iotmonitor.cloud> としています。実際のリクエストには他のフォーム・パラメータも含



まれているが、読みやすくするため、ここでは省略しています。

コードからわかるように、エンドポイントに HTTP で接続するために **RestSource** を作成し、パス、パラメータ、接続設定（タイムアウト値など）といった一般的な情報を指定しています。次に、**RestSource** を **ObjectDataProvider** に渡します。**ObjectDataProvider** はリクエストを作成し、結果を **answer** という変数名のプロパティに格納します。そして、**ExecutorService** インスタンスを **DataFX** の **ObjectDataProvider** に渡します。これは、大量のログ情報を送信しなければならない場合

もあることを想定しているためです。すべての通信に1つのスレッドを使用すると転送が遅延する可能性があります。各メッセージに1つのスレッドを使用すると、大量のリソースが消費されてしまう可能性があります。そのため、**ExecutorService** には通常5つのスレッドが割り当てられます。

**DataFX** の詳細な説明は、本記事では割愛します。詳細は、

<http://datafx.io> を参照してください。

インターネット接続されたデバイスで多発する問題のひとつは、インターネットとの接続が切断されてしまうことです。当然ながら、接続が切断されたというログ・メッセージは送信できません。そのような場合は、**MonitorHandler** がすべてのログ・メッセージをファイル・システムに格納します。接続が回復すると、メッセージが IoT モニター・クラウドに送信されます。

### JavaFX モニターのクライアント・アプリケーション

JavaFX モニターのクライアント・アプリケーションは、IoT モニター・クラウドに接続し、オペレータが参照できるモニタリング情報を表示します。IoT モニター・クラウドでは、登録オペレータのリストが管理されています。また、オペレータは1つまたは複数のカード・システムに関連付けられています。カード・システムとは、ひとまとまりとなった CID のグループです。このグループは通常、ある都市のすべての CID です。

そのため、IoT モニター・クラウドの設定において、認証は重要な要素です。図3は、JavaFX モニター・クライアントのログイン画面です。

オペレータが認証されると、監視コンソール（図4参照）が表示されます。

コンソールの左上には、オペレータがアクセスできるカード・システムのリストが表示され、その下には、アクセス可能なカード・システムに

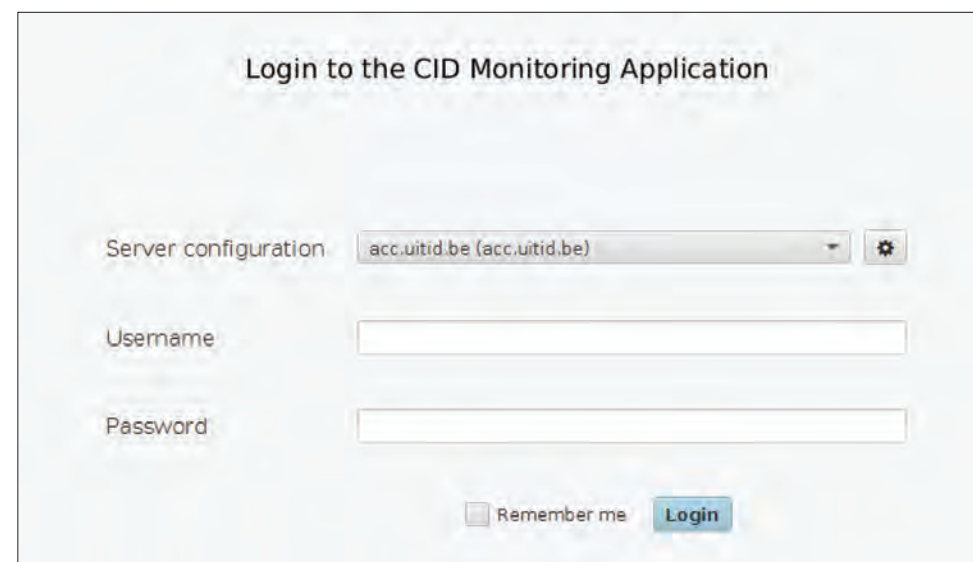


図3

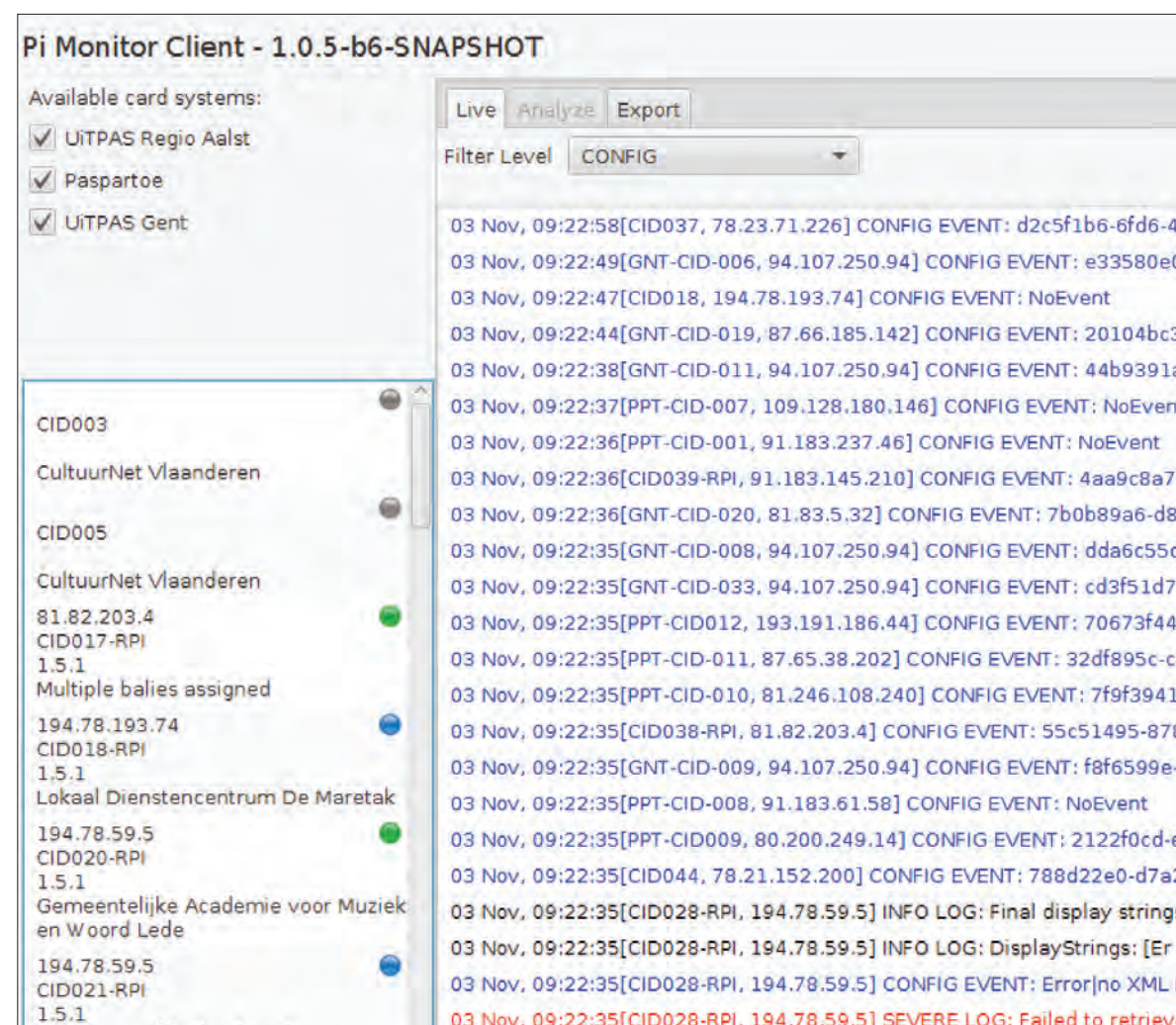


図4

### モニタリング

**Java EE 7 API** を使用すると、高い性能とスケーラビリティを両立し、クライアントとしてのニーズ（WebSocket のサポートなど）にも対応できる**クラウドベースのモニタリング・システム**の開発を行うことができます。



登録されている CID が表示されます。コンソールの中核部分に表示されているのがモニタリング・データで、オペレータが管理する各 CID が生成したものです。

通常、表示されるデータは膨大な量になります。そのため、アプリケーションにはさまざまなセレクトやフィルタが組み込まれており、オペレータがデータの特定の部分を選択できるようにになっています。たとえば、次のようなものがあります。

- 特定期間内の情報だけを表示する時間セレクト
  - 特定のレベルやレベルの範囲でメッセージを選択するレベル・セレクト
  - ログの分析対象とする CID を1つまたは複数選択する CID セレクト
- 左側のリストから特定の CID をクリックして選択すると、そのモニタリング・データが表示される。CID が選択されていない場合は、すべての CID のデータが表示される

データを分析する際に、通常、オペレータはセレクトをひんぱんに切り替えます。セレクトを変更するたびにバックエンドの呼出しを行うと、アプリケーションの応答性が悪くなります。しかし、JavaFX を Java 8 の機能と組み合わせることで、オペレータがデータ・セットをクライアント上で操作できるようになります。アプリケーションが起動すると一部のデータが取得されます。さらにデータが必要になると、データがアプリケーションに読み込まれ、必要

なくなったデータはガベージ・コレクションの対象となります。この仕組みによって、アプリケーションを実行しているクライアント・システムのメモリや処理能力を有効に活用できます。一般的に、JavaFX モニター・アプリケーションはリスト上に表示されているデータよりもはるかに多いデータ・エントリをメモリに保持しています。

コンソールの中核部分であるリスト・コンポーネントは、次のように定義されています。

```
private ListView<Event>
eventsListView
```

この `ListView` に表示される項目は、`sortedEvents` という変数名を持つ JavaFX の `SortedList` に保持されています。`sortedEvents` は、次のようにして `ListView` と関連付けられています。

```
eventsListView.setItems(
sortedEvents);
```


IoT モニター・クラウドから取得したモニタリング項目のすべてのデータは、`events` という変数名を持つ `Event` 型の `ObservableList` に格納されます。

```
ObservableList<Event> events;
```

新しいイベントは、DataFX が **リスト 4** と同様のコードによってバックエンドから取り込むか、後述するように IoT モニター・クラウドから JavaFX モニター・アプリケーション

リスト5 / リスト6 / リスト7

```
FilteredList<Event> filteredEvents = events.filtered (e → true);
```

 [すべてのリストのテキストをダウンロード](#)

に WebSocket 経由で送信されます。いずれの場合でも、イベントは `ObservableList` に格納されているため、`ListView` がすべてのデータを使用してイベントを表示できます。表示内容には、`ListView` が作成された後で追加されたイベントも含まれます。

なお、`events` という変数名の `ObservableList<Event>` に格納されているのは生のイベント・データですが、`ListView` に渡されるイベントは `sortedEvents` という変数名の `SortedList<Event>` に格納されています。この2つのリストには関連があります。まず、生のイベント・データのリストは、さまざまなセレクトに一致するイベントのみを含むようにフィルタリングされます。この実現のため、フィルタリング機能を提供する Java 8 Stream API を使用しています。また、`FilteredList<Event>` 型の中間フィールド `filteredEvents` を作成します。このフィールドの初期設定を **リスト 5** に示します。

`FilteredList` は、JavaFX 8 で追加されたクラスです。このクラスは、元のリストに基づきつつ、predicate として指定されたフィルタ条件も考

慮して要素のリストを管理する非常に強力なアプローチを提供しています。

この初期化の仕方から、最初はすべての生のイベント・データが `filteredEvents` リストに含まれていることがわかります。セレクトが変更されると、フィルタ条件も変わります。

```
filteredEvents.setPredicate(
validEvent())
```

`validEvent()` は、イベントを `filteredList` に含めるかどうかをチェックする条件を返します。

今回の例では、イベントが許容されるためには、前述の3つのセレクトによって適用される境界条件を満たしている必要があります。`validEvent()` メソッドの実装を **リスト 6** に示します。

このメソッドの `return` 文には、それぞれが条件を返す3つのメソッドが含まれています。ここでは、`isShowLevel()` 条件のコード (**リスト 7** 参照) を見るだけにします。他の関数も、少し複雑になっています。が同様の方法で作成されています。`isShowLevel()` 条件では、与えられ



たイベント `e` のログ・レベルが、セレクトタによって選択されたレベルと同じまたはそれ以上であるかどうかをチェックしています。

これで `ListView` に `filteredEvents` が表示されるはずですが、イベントが表示される順番は保証されません。この問題は、`Event` クラスに `java.util.Comparable` インタフェースを実装することで解決できます。このインタフェースを、直近のイベントが常に最初に表示されるように実装します。JavaFX 8 には `SortedList` クラスが追加されており、`ListView` インスタンスを `Comparator` に基づいてソートできるようになっています。この機能を次のように使用します。

```
SortedList<Event> sortedEvents =
    filteredEvents.sorted();
```

`ListView` に渡されるのは、`ObservableList` を拡張したこの `SortedList` です。このように、Java 8 の機能を使用すると、ソートされていない大量の生データから関連するエントリのみを取り出し、ソートして表示することが簡単にできます。

## WebSocket

図 4 の時間セレクトタには、「Live」というタブがあります。このタブで、リアルタイム監視が可能です。

多くの場合、デバイスのリアルタイム監視は非常に重要です。リアルタイム監視によって、小さな問題が大きな問題に発展することを未然に防げる可能性があるためです。また、

ユーザーがオペレータに連絡して問題を報告するのではなく、オペレータ自身で問題を検知することが容易になります。そのため、IoT モニター・クラウドには、システムにデータ・エントリが到着した際に即時送信する機能があります。

IoT モニター・クラウドと JavaFX モニター・アプリケーションは、両方とも Java API for WebSocket (JSR 356) を使用しています。ユーザーが JavaFX モニター・アプリケーションに正常にログインすると、アプリケーションは IoT モニター・クラウドとの間で即座に WebSocket 接続を確立します。前述の DataFX フレームワークには、WebSocket コンポーネントも含まれています。このコンポーネントは JSR 356 のクライアント部分を使用しており、IoT モニター・クラウドへの WebSocket 接続を開きます。IoT モニター・クラウドは、Java EE 7 のリファレンス実装である GlassFish 上で実行されています。そのため、IoT モニター・クラウドには、最初から JSR 356 の実装が含まれています。したがって、IoT モニター・クラウドに WebSocket エンドポイントを登録するのは非常に簡単です。IoT モニター・クラウドは、新しく受信したすべてのイベントをその情報の参照権限を持つクライアントにブロードキャストします。

## まとめ

本記事では、現場の IoT デバイスを監視できることの重要性について説明しました。大量のデータを

生成する多数のデバイスを使用する場合、とりわけ、権限の異なるさまざまなグループのユーザーが存在する場合は、クラウドベースのソリューションによるデータ収集が理想的です。Java EE 7 API を使用すると、高い性能とスケーラビリティを両立し、クライアントとしてのニーズ (WebSocket のサポートなど) にも対応できるクラウドベースのモニタリング・システムの開発を行うことができます。

モニタリング・データは、JavaFX を使用して表示しています。Java 8 API は、さまざまな条件で関連データのフィルタリングやソートを行う際に活用できます。

組込みデバイスでは Oracle Java SE Embedded が、IoT モニター・クラウドでは GlassFish 4.1 のインスタンスが実行されています。監視クライアントは JavaFX 8 で作成した自己完結アプリケーションです。これはまさに、「Java everywhere」のよい実例だと言えるでしょう。</article>

MORE ON TOPIC:



## LEARN MORE

- [DataFX](#)
- [JSR 356](#)

# CREATE THE FUTURE

[oracle.com/java](http://oracle.com/java)



ALEXANDER  
BELOKRYLOV

## Alexander Belokrylov :

Oracle Java ME Embedded の Principal Product Manager。活動拠点はロシアのサンクトペテルブルク。エンジニアとしてキャリアを開始してから 15 年以上にわたって IT 業界に従事。サンクトペテルブルクの [JUG.ru](#) コミュニティの共同創設者で、Java や組み込みデバイスに意欲的に取り組んでいる。

# マイクロコントローラでの Oracle Java ME Embedded 8 の使用

# Oracle Java ME Embedded の Device I/O API で周辺デバイスにアクセスする

# 本記事では、Oracle Java ME Embedded 8.1 と Device I/O API、ARM Cortex-M4 プロセッサ・ベースの Freescale FRDM-K64F ボードを使用して、DS1621 センサーからデータを取得する方法を説明します。DS1621 センサーは、Inter-Integrated Circuit (I<sup>2</sup>C) バス・インタフェースを搭載したデジタル・サーモメータおよびサーモスタットです。

本記事では、次の点について説明します。

- Freescale FRDM-K64F ボードの Java 機能の有効化
- DS1621 センサーの読み書きに必要なプロトコル情報の取得
- Device I/O API と Java ME SDK ツールを使用して I<sup>2</sup>C デジタル・サーモメータにアクセスする Oracle Java ME Embedded アプリケーションの開発

# Oracle Java ME Embedded 8.1 の概要

Oracle Java ME Embedded は、ARM Cortex-M3 や Cortex-M4 マイクロコントローラ・ベースのデバイスなど、リソースに制約がある組み込みデバイス向けに最適化された Java ランタイムです。Oracle Java ME Embedded を使用すると、いくつかのメリットが得られます。

最初に挙げられるのが、「Write once, run anywhere (一度書けばどこでも実行できる)」の原則です。いくつかの制限があるものの、現在断片化している組み込みシステムのハードウェアやソフトウェアを抽象化し、一貫性があり機能の豊富な Java プログラミング層を提供することで、この原則を組み込みデバイスの世界に適用できます。

次に、Oracle Java ME Embedded プラットフォームに統合されている Application Management System によって、

アプリケーションの配置、アップデート、開始、停止、削除が可能になる点です。

3 番目は、シリアル・ペリフェラル・インタフェース (SPI) バス、I<sup>2</sup>C、ユニバーサル非同期レシーバ/トランスミッタ (UART)、汎用 I/O (GPIO) ピンなどを使用する周辺機器に、Device I/O API を使用してアクセスできることです。

最後に、Oracle Java ME Embedded を使用すると、無料で強力な開発ツールを取得できるほか、さまざまな標準サービスや API を使用できます。使用できる API として、ファイル I/O ([JSR 75](#))、ワイヤレス・メッセージング ([JSR 120](#))、Web サービス ([JSR 172](#))、セキュリティと信頼性のあるサービス ([JSR 177](#))、ロケーション ([JSR 179](#))、XML ([JSR 280](#))、[JSON](#)、[OAuth 2.0](#) があります。

2014 年 11 月にリリースされた最新バージョンの Oracle

Java ME Embedded プラットフォームでは、2つの大きな機能拡張が行われました。1つ目は、mbed OS を搭載した Freescale FRDM-K64F ボード向けにポートを設け、RAM が非常に限られる ARM Cortex-M3 と Cortex-M4 マイクロコントローラをサポートした点です。

2つ目は、Eclipse 統合開発環境 (IDE) のサポートです。Eclipse の Java ME SDK プラグインは、開発の際に Oracle エンジニアの意見が多く取り入れられた Mobile Tools for Java 2.0 の最新リリースをベースとしています。この機能拡張によって、オンデバイス・デバッグ、メモリやネットワークのモニター、CPU やメモリのプロファイリングなど、Java ME SDK の数々のすばらしい機能が Eclipse で使用できるようになりました。Java ME SDK 8.1 Eclipse Plugin ファイル をダウンロードしてインストールすると、このような機能を試す





今回は、すべてのピンを VDD ピンに接続します。

ドキュメントによると、DS1621 のアドレスは 7 ビットで構成されます。

- 最初の 4 ビットは「1001」
- 最後の 3 ビットは、A2、A1、A0 ビットの論理状態

したがって、今回の設定では、アドレスは「1001111」、つまり「0x4f」になります。

データ・シートの7ページに、DS1621 センサーは通常モード(クロック・レート 100kHz) または高速モード(クロック・レート 400kHz) で動作できることが記載されています。今回は通常モードを使用します。そのため、周波数は 100,000Hz になります。

温度測定を行うためには DS1621 センサーの設定が必要で、設定レジスタ (0xAC) に適切なビットを書き込まなくてはなりません。データ・シートの 5 ページに、設定ビットの定義が記載されています。最後のビットの Lsb に注目してください。このビットを 0 に設定すると、DS1621 センサーは継続的に温度を測定し、結果を温度レジスタに記録します。そのため、設定レジスタの値は「0x00」に設定する必要があります。

データ・シートの 10 ページには、センサーのコマンド・セットが記載されています。温度測定を開始するためには、DS1621 センサーに「Start Conversion T」(0xEE) コマンドを送信し、測定した温度をデジタル値に変換する必要があります。

同じページに、温度測定結果の読み取りについて記載されています。セン

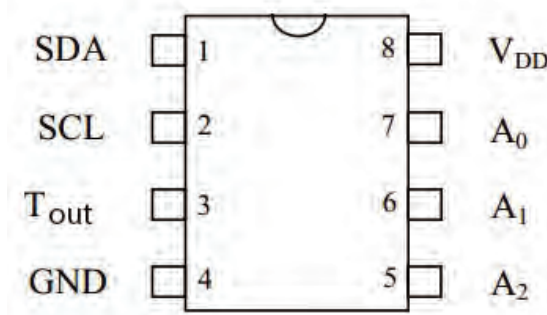


图1

サーに「Read Temperature」(0xAA) コマンドを送信する必要があり、送信後、センサーは2バイトのデータを送り返します。1バイト目は、摂氏温度です。2バイト目の先頭ビットは、0.5度分の追加を示します。今回は、温度の分解能は1度とします。そのため、最初のバイトのみを読み取ります。

**表 2** に、本セクションで学んだことをまとめます。

## Oracle Java ME Embedded アプリケーションの開発

ここからは、温度を測定する Oracle Java ME Embedded アプリケーションの開発に入ります。

まず、お使いの IDE で新しい Java ME プロジェクトを作成します。なお、Oracle Java ME Embedded アプリケーションは Java SE アプリケーションとは異なります。Oracle Java ME Embedded アプリケーションでは、`MIDlet` クラスを拡張し、`startApp()` および `destroyApp()` という 2 つのメソッドをオーバーライドします。機能は `MIDlet` の外に出すことが推奨されています。そのため、`DS1621` クラスを作成します。

Oracle Java ME Embedded アプ

ピン	記号	説明
1	SDA	2 線式シリアル通信ポート用のデータ入出力ピン
2	SCL	2 線式シリアル通信ポート用のクロック入出力ピン
3	TOUT	サーモスタット出力。温度がユーザー定義の上限温度 (TH) を超えるとアクティブになり、ユーザー定義の下限温度 (TL) を下回るとリセットされる
4	GND	グラウンド・ピン
5	A2	アドレス入力ピン
6	A1	アドレス入力ピン
7	A0	アドレス入力ピン
8	VDD	電源電圧入力ピン

表1

項目	値
I <sup>2</sup> C バス・アドレス	0X4F
クロック・レート	100,000HZ
設定レジスタ (0XAC) に書き込む値	0X00
温度変換を開始するコマンド	0XEE
温度測定結果を読み取るコマンド	0XAA

表2

リケーションから I<sup>2</sup>C デバイスにアクセスするためには、Device I/O API の `jdk.dio.i2cbus` パッケージにある `I2CDevice` クラスと `I2CDeviceConfig` クラスを使用します。最初に、DS1621 センサーのデータ・シートを使用し、`I2CDeviceConfig` クラスのインスタンスを作成する必要があります。次のコードをご覧ください。

```
cfg = new I2CDeviceConfig(BUS_ID,  
CFG_ADDRESS,  
ADDRESS_SIZE,  
FREO);
```

ここでは、**リスト1**に示すコードを使用します。そうすると、**リスト2**のように **I2CDevice** クラスのインスタンスを作成できます。

I<sup>2</sup>C インタフェースへの書込みを簡単に行えるように、`write` メソッドを作成します (**リスト 3** 参照)。実際のコード



 すべてのリストのテキストをダウンロード