

# Java™ magazine

By and for the Java community



## 02 クラウドを 自在に操る

オラクルのCameron Purdyが語る - クラウド・コンピューティング・プラットフォームの標準技術となるJava EEの進化 -

## 09 JAVAFXと SWING

Swingアプリケーションの移行: アプリケーション全体を書き直すことなくリッチなJavaFX機能を活用

## 13 軽量パブリッシュ/ サブスクライブ

Java EE 6とCDIを利用して無駄のない効率的なローカル・パブリッシュ/サブスクライブ通信を実装

## 17 GLASSFISHの クラスタリングと 高可用性

アプリケーションの水平スケーラビリティを実現

# TAME THE CLOUD

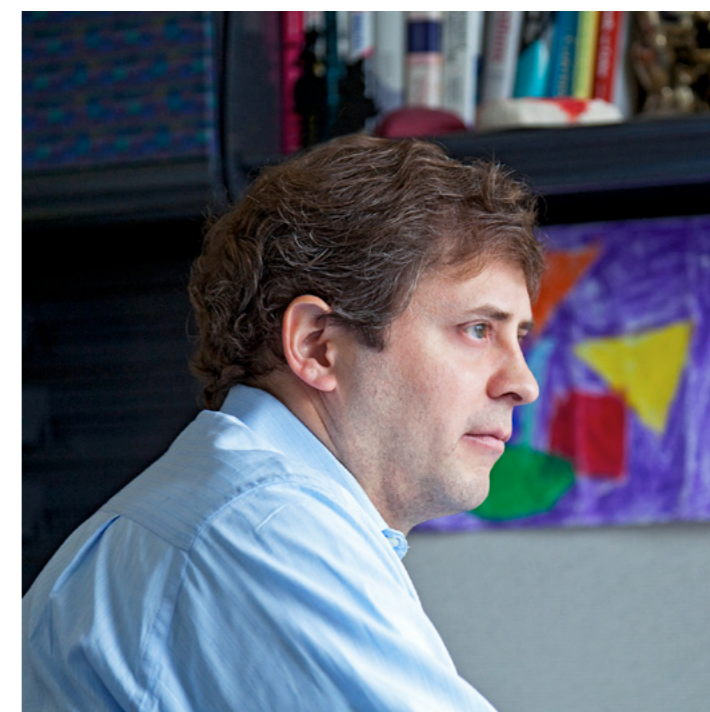
クラウド・コンピューティング・プラットフォームの  
標準技術となるJava EEの進化

# Cloud/Java EE

# クラウドを自在に操る

オラクルのCameron Purdyが語る - クラウド・コンピューティング・プラットフォームの標準技術となるJava EEの進化 - **MICHAEL MELOAN**

# インターネット情報システムの世界は、以前よりより多くの情報を取り込む事によって、より複雑化しています。誰もが重要視することは、スタンドアロン・アプリケーションを、動的な共有環境に移行し、高いスケーラビリティ、自動化、効率化を進めることです。クラウド・コンピューティングはJava EEにとって変革技術になる事をお約束します。ここでは、オラクルのJava EE開発部門の統括責任者であるCameron Purdyが、クラウド・コンピューティング対応における主要課題や、エンタープライズ開発者と顧客に対する影響について概要を語ります。



**Java Magazine:** エンタープライズ環境にとって、クラウド・コンピューティングの利点とは何でしょうか。

**Purdy:** 現在、クラウド・テクノロジーの開発は比較的初期の段階にあります。おもに注目されているのはPaaS (Platform as a service) が持つ可能性です。数々のインフラストラクチャ・クラウド・ベンダーが、eコマース・サイトで買い物をするかのように、いとも簡単に何十台、あるいは何百台ものサーバーを提供しています。このようにITインフラストラクチャは分散化、一般化に向かって進んでいます。開発環境やミドルウェア・コンポーネントにおいて、ハードウェアとソフトウェアの各要素が抽象化される結果、開発者はエンタープラ

写真: DAVE BRADLEY  
画像: I-HUA CHEN





Oracle WebLogic Server 12cについて話すオラクルのMike LehmannとArun Gupta。ブラジルのサンパウロで開催されたOracle OpenWorld Latin America 2011にて

また、伸縮性という概念を取り入れることで、可用性も確保しやすくなります。アプリケーションは、1台のサーバーが保守や障害のために停止しても継続して運用を続けられるような十分な堅牢性が必要です。またアプリケーションは、同時実行ユーザーの増加や、急な負荷の増加によって、処理能力の追加が必要な場合にも迅速に対応する必要があります。そのため、伸縮可能なアプリケーションはリソースを動的に追加できるようにしていなければなりません。この点はクラウド・コンピューティングのメタファーに非常に良く適合します。

可用性や伸縮性の概念は、PaaSやIaaS (Infrastructure as a service) 上に構築されたエンタープライズ・アーキテクチャと共にあるべきものです。

マルチテナント対応も、クラウド・コンピューティングの重要な概念です。マルチテナントとは、単一環境上で複数のユーザーを並行して稼働させる機能です。たとえば、CRM (顧客管理) システムの場合、数多くの企業に対してサービスを提供する必要がありますが、それぞれの企業が異なる1つのテナントとして動作します。そして、複数のテナントを専用のハードウェアや仮想化環境でサポートして、1台のハードウェア・インフラストラクチャ内の1つの仮想マシン、もしくは複数の仮想マシンを各テナントに割り当てることができます。マルチテナントにはさまざまなレベルがあり、単一のJVM (Java仮想マシン) 内部に複数のテナントを配置するような小さな単位のものもあります。

マルチテナント・アーキテクチャでは例外なくトレードオフが発生します。真っ先に挙げられるのが、セキュリティ、リソース利用率、コストの問題です。現在、JVM内部レベルで隔離、保護を行う事を重要と考え、その調査に多大な投資を行っています。また、特定のテナント毎に使用する、CPUの使用状況やメモリ使用状況を測定する方法についても調査しています。テナントで消費するリソースをどう管理し制限するかが、マル

チテナント・システムの根本的な課題です。1つのテナントがCPUを使い果たすことのないように、あるいはシステムを破壊するほど大量のメモリを使用することがないように管理しなければなりません。オラクルが目指しているのは、利用者に対してこれらの課題を管理する基盤技術を提供することです。

Java EEの各方面を見直す中で、パートナーと将来のJava EEプラットフォームの計画を立てる際に、マルチテナント、可用性、伸縮性のすべてが重要な検討課題になるでしょう。

**Java Magazine:** Java EE 7リリースの新機能について教えてください。

**Purdy:** オラクルは現在、業界パートナーと緊密に連携しながらJava EE 7の仕様策定や提供に向けて取り組んでいます。Java EE 6の大きなテーマであった「開発容易性」はJava EE 7でも引き続き重

要ですが、Java EE 7のもっとも大きなテーマはクラウド対応です。Java EEのコンテナベース・アーキテクチャと本来備わっているリソース・アクセスの抽象化は、クラウド・コンピューティングに非常に適しています。このアプローチによって、プログラミング・モデルに大きな変更を加えることなく、移植可能なアプリケーションを1台のマシンに配備することも、大規模なクラスタ環境に対してインストール・配備することも可能になります。クラウド・コンピューティングに向かうことで、Java EE 7はこの強力なパラダイムのさらなる進化につながるはずです。

マルチテナントは、仮想サーバーがリクエストをテナントにマッピングするという手法で実現する予定です。また、CDI (Contexts and Dependency Injection)、JPA、JAX-RS (Java API for RESTful Web Services)、JSF、サーブレッ



オラクルのアーキテクトであるGene Gleyzerから担当プロジェクトの状況について報告を受けるPurdy。マサチューセッツ州バーリントンにあるオラクルのオフィスにて



ソフトウェア・エンジニアの  
Dave Carranoの作業内容  
を確認するPurdy(上)。Java  
EE 7について話す、開発部門  
の統括責任者Alex Gleyzer  
とPurdy(右)



ト、EJBにもより重点を置いています。

さらにJava EE 7では、WebSocketやHTML5、JAX-RS 2.0で定義している最新のHTTPクライアントAPIなどのWebの最新テクノロジーも新たにサポートする見込みです。

さらにJava EE 7では、WebSocketやHTML5、JAX-RS 2.0で定義している最新のHTTPクライアントAPIなどのWebの最新テクノロジーも新たにサポートする見込みです。

**Java Magazine:** Java SE 8ではモジュール化とラ

ムダ式が大きなテーマとなっていますが、Java EEにとってこれらの開発にはどのような利点がありますか。

**Purdy:** これらの2つの機能は非常に大きな可能性を秘めています。モジュール化はJava EEを通じてオラクルにとって重要な分野であり、スケーラビリティ、堅牢性、保守性の利点をもたらすものです。Java SEの標準モジュール・システムの設計と実装については、[Project Jigsaw](#)のもとにOpenJDKで検討しています。Java SE 8のモジュール化実装では、Javaエンタープライズ・プラットフォームの進化に役立つ土台となる機能を提供する予定です。

これまですでに  
OSGi (Open Services  
Gateway initiative) の  
成果を活用して、Oracle  
GlassFish Serverおよび  
Oracle GlassFish Server  
Open Source Editionにモ  
ジュール化を取り入れてい  
ます。Project Jigsawの成  
果物であるモジュール・シ  
ステムがJava SEに導入さ  
れる事が決まりましたが、  
今後もOSGiなどの標準技  
術と共に、このアプローチ  
への強化を進め、より密接  
に検討した結果をJVMサポ  
ート機能として統合し、提供

していく予定です。

次にラムダ式についてですが、これは一般的に、集計処理の実装等で役立つ機能です。集計処理を行う際、マルチコア・プロセッサを頻繁に利用して並列処理を行うことができます。これまでの調査で、JDBC、JPA、EJB、JMSフィルタリングなどの多くの技術領域に対してラムダ式を適用できることがわかっています。ラムダ式はJava以外のプログラミング言語環境でも優れた効果を発揮しているため、Java EEでラムダ式をぜひサ

ポートしたいと考えています。

**Java Magazine:** 最後に、エンタープライズ・システムの未来にとってクラウドがどれほど重要なのか、見解を聞かせてください。

**Purdy:** 現在は、新しいイノベーションの波が押し寄せているところだと感じています。15年ほど前にWeb指向のテクノロジー開発が躍進したときも同じような状況でした。また10年前にはJ2EEのリリースに伴って、Java EEに移行するビジネス・アプリケーションが急増しました。今もまた同じことが起きています。コスト削減や柔軟性の観点からクラウド・テクノロジーを有効活用しようと、信じられないほど多額の投資が行われています。

同時にこの業界は、HTML5によってもたらされる非常に高度なユーザー・エクスペリエンスにも向かっています。このように重要で新しい技術が組み合わされることで、Java EEはさらに進化し、新しいパートナー、組織、開発者がこの分野に参入するまたとないチャンスが生まれます。その結果、テクノロジーの展望や、エンタープライズ市場全体が大いに拡大することでしょう。

HTML5やクラウド・テクノロジーの可能性模索するときに、さまざまなレベルのすばらしいイノベーションに出会えることを私は楽しみにしています。そのようなイノベーションに関われるのは、本当にわくわくすることですから。

&lt;/article&gt;

## Michael Meloan

IBMメインフレームとDEC PDP-11のアセンブリ言語のコーディングに関するプロとしてキャリアをスタートさせ、PL/I、APL、C、Javaのコーディング分野でも高い専門性を備えている。また、WIRED、BUZZ、Chic、LA Weeklyに

**LEARN MORE**

- [Java EE at a Glance](#)
- [Oracle WebLogic Server](#)



ARUN GUPTA



## Cloud/Java EE

# Java EE 7を見据えて

# Java EE 7 JSRについて学ぶ

2012年後半に予定されているJava EE 7のリリースを前に、リリースが計画されているJSRの動きが活発化しています。Java EE 7リリースはクラウドに移行する業界の最新ニーズを反映する予定です。リリース時期が決まっています。そのため、準備が間に合わない機能については、Java EE 8に持ち越される予定です。

ここでは、Java EE 7のさまざまな仕様に関して、おもな機能の最新情報や概要を説明します。

## Java EE 7 仕様(JSR-342)

- **メイン・テーマ: プライベート・クラウド、もしくはパブリック・クラウド上で容易なアプリケーションの実行**
- **マルチテナント、リソース共有、サービス品質、アプリケーション間の依存性などPaaS実行環境を表すためのアプリケーション・メタデータ記述子を定義**
- **HTML5、WebSocket、JSONなど最新標準技術を採用し、それぞれ標準APIを提供**
- **マネージドBean、EJB、サーブレット、JSF、CDI、JAX-RS間の不整合を解消**
- **Web ProfileにJAX-RS 2.0を追加予定。JMS 2.0 APIの修正予定**

- 既存のいくつかのテクノロジーを更新予定。Java EE ([SR-236](#)) および JCache ([JSR-107](#)) 用の Concurrency Utilities を追加予定
- ステータス:
  - JCP の承認済み
  - 仕様リード: Linda DeMichiel, Bill Shannon (オラクル)
  - [プロジェクトのページ](#)
  - [メーリング・リスト・アーカイブ](#)  
[jsr342-expert@javaee-spec.java.net](mailto:jsr342-expert@javaee-spec.java.net), [users@javaee-spec.java.net](mailto:users@javaee-spec.java.net)

## Java Persistence 2.1 (JSR-338)

- マルチテナントのサポート
- ストアド・プロシージャやベンダー関数のサポート
- クエリの更新/削除
- スキーマ生成のサポート
- Persistence Context の同期
- リスナーへのCDIインジェクション
- ステータス:
  - JCPの承認済み
  - 仕様リード:Linda DeMichiel (オラクル)
  - プロジェクトのページ
  - メーリング・リスト・アーカイブ、  
[jsr338-experts@jpa-spec.java.net](mailto:jsr338-experts@jpa-spec.java.net)、[users@jpa-sepc.java.net](mailto:users@jpa-sepc.java.net)

## JAX-RS 2.0: The Java API for RESTful Web Services (JSR-339)

- クライアントAPI:ビルダー・パターンを使用した低レベルAPIの導入、その上位レベルのAPI導入の可能性もあり
- ハイパーメディア:容易に、リソースに関連付けられたリンクの作成と処理が可能
- Beanバリデーションを使用したフォームやクエリのパラメータ検証
- より緊密な@Injectとの統合
- サーバー・サイドの非同期リクエスト処理
- "qs"を使用したサーバー・サイドのコンテンツ・ネゴシエーション
- ステータス:
  - JCPの承認済み、[初期ドラフト公開](#)、[Javadocドラフト作成](#)
  - 仕様リード: Santiago Pericas-Geersten、Marek Potociar (オラクル)
  - [プロジェクトのページ](#)
  - [メーリング・リスト・アーカイブ](#)、[jsr339-experts@jax-rs-spec.java.net](mailto:jsr339-experts@jax-rs-spec.java.net)、[users@jax-rs-spec.java.net](mailto:users@jax-rs-spec.java.net)

## Java Servlet 3.1 仕様 (JSR-340)

- Webアプリケーションに対するPaaSモデルの最適化

- セキュリティ、セッション、リソースに対するマルチテナント対応
- NIO.2ベースの非同期I/O
- 非同期サブレットの簡略化
- Java EE Concurrency Utilitiesの活用
- WebSocketのサポート
- ステータス:
  - JCPの承認済み
  - 仕様リード: Shing-Wai Chan、Rajiv Mordani (オラクル)
  - [プロジェクトのページ](#)
  - [メーリング・リスト・アーカイブ](#)、[jsr340-experts@servlet-spec.java.net](mailto:jsr340-experts@servlet-spec.java.net)、[users@servlet-spec.java.net](mailto:users@servlet-spec.java.net)

## Expression Language 3.0 (JSR-341)

- ELContextを解析コンテキストと評価コンテキストに分割
- カスタマイズ可能なEL強制ルール
- EL式の静的メソッドとメンバーの直接参照
- 同値、文字列連結、サイズ判定などの演算子を追加
- 式評価前/評価中/評価後のイベント生成など、CDIとの統合
- ステータス:
  - JCPの承認済み
  - 仕様リード: Kin-man Chung



**Concurrency Utilities for  
Java EE (JSR-236)**

このJSRは長年検討が続いているものですが、Java EE 7のリリースまでに再開し完了する予定です。

- JSR-166 を基礎として明快でシンプルな他に依存しないAPIを提供、Java EEコンテナでの利用に合わせて最適化
- パッケージ: javax.util.concurrent
- ステータス:
  - JCPの承認済み
  - 仕様リード: Anthony Lai, Naresh Revanuru (オラクル)
  - Project page: TBD
  - メーリング・リスト・アーカイブ: 未定

**Java API for JSON Processing  
(JSR-353)**

「Java API for JSON Processing」(JSON処理用のJava API)のJSRが、JSR-353として提出されました。

- ステータス:
  - JCPの承認済み
  - 仕様リード: Jitendra Kotamraju (オラクル)
  - プロジェクトのページ
  - メーリング・リスト・アーカイブ

すでにほとんどのJSRに対して専門家グループ(EG)を編成済みですが、公開エイリアスに登録してドラフトをレビューするという手段でJSRに参加していただけます。オラクルが主導するJSRは、Java.netの専用プロジェクトで透明性

を保ちながら進行しています。たとえば、EGの公開ディスカッションにアクセスし、新たに投票されたJCP 2.8ルールが定める要件を確認できます。

JCP 2.8の後のすべてのJSRが、より高い透明性を保って進行しています。JCP 2.8の特に重要な話題について、いくつかで紹介しましょう。

- EGメンバー名を公開
  - EGの作業状況を公開エイリアスで報告
  - スケジュールを公開し、最新情報を定期的に提供
  - 誰でもWikiを参照し書き込むことで、これまでのステータスに関する考察が可能
  - jcp.orgでディスカッション・ボードを提供
  - 読取り専用の公開課題トラッカーを提供
- 今後これらの機能のすべてをGlassFish (Java EE 7のリファレンス実装)に統合します。

&lt;/article&gt;

**LEARN MORE**

- [Java EE Platform 仕様](#)
- [GlassFish](#)

COMMUNITY

JAVA IN ACTION

JAVA TECH

ABOUT US



08

**Java Magazine  
日本版 始動!**

Oracle Corporation の公式技術者支援サイト、Oracle Technology Network (OTN) より、Java 技術者・開発者向けのデジタル・マガジン、『Java Magazine』が創刊されました。この Java Magazine の記事から、日本のエンジニアの方々に特に関心の高いコンテンツをピックアップした日本語バージョンを、『Java Magazine 日本版』として隔月でお届けします。エンタープライズシステムから、組み込み系まで、最新の Java テクノロジーに関するトピックスを幅広くカバーし、Java 活用事例などの業界最新トレンドをわかりやすくお伝えします。

**OTN Newsletter にて、Java Magazine 日本版の  
最新の発刊をご案内します。  
今すぐ、ニュースレターをご購読ください! (登録無料)**

OTN Newsletter の登録はこちらから

<http://oracle.co.jp/joinotn>

(※すでに Oracle.com (MyOracle) アカウントをお持ちの方は、プロフィール情報の更新でニュースレター購読希望を追加してください。)



**第1号** [2011年11月発行] 【創刊記念号】

- Java 利用の成功例
- Java 入門
- (最初の一步・RESTful Web サービスの概要)
- お待たせしました! Java 7 の登場
- JDK 7 コーディング方法の変革 ほか



**第2号** [2012年1月発行]

- 「進化しつづける Java」 (JavaOne 2011 レポート)
- JavaFX 2.0 の主要機能
- JavaFX 2.0 により再活性化するクライアントサイドの Java 開発 ほか



**第3号** [2012年3月発行]

- クラウド・コンピューティング・プラットフォームの標準技術となる Java EE の進化
- リッチ・クライアント: JavaFX と Swing の統合
- エンタープライズ Java / 軽量パブリッシュ / サブスクリプション
- GlassFish で実現するシンプルなクラスタリングと高可用性 ほか



Oracle Technology Network 日本語サイトでは、Java をはじめとする、最新の Oracle ソフトウェアや各種ツールの無料ダウンロード、ドキュメント、技術解説記事、オンラインセミナーなどをご提供しています。是非、ご利用ください。

<http://www.oracle.com/technetwork/jp/>



## パート2

## Swingアプリケーションの移行： アプリケーション全体を書き直すことなくリッチなJavaFX機能を活用する

この記事ではパート1  
を踏まえ、リッチなJavaFX  
機能を活用するための  
Swingアプリケーションの  
移行方法について説明し  
ます。この方法を利用す  
れば、アプリケーション全  
体を書き直す必要はあり  
ません。ここでは、JavaFX  
の興味深い機能をいくつ  
か利用して、アプリケー  
ションのルック・アンド・  
フィールを改善する方法  
を学ぶために、サンプル  
のSwingアプリケーションのテーブ  
ル・コンポーネントを置き換えます。

きます。

アプリケーションは複数のパッケージに整理されているので、関連のある部分を簡単に見つけ出すことができます。ここでは、**StocksMonitorMainWindows** クラス

とQuote クラスを修正し、さらに QuoteTable クラスを新しい QuoteFXTable クラスに置き換えます。

単純化のため、QuoteFXTable クラスは、既存の QuoteTable クラスと完全に互換なものにすることを目標にします。このため、このクラスはObserver インタフェースと PropertyChangeListener インタフェースを実装し、正確なシングネチャの setQuoteTable メソッドを持つ必要があります。

APIを利用できます。このAPIは、セルの内容をカスタマイズするためのセル・ファクトリや、ユーザーによる実行時の列の並び替え、複数の列をグループ化する列のネスト化などの機能をサポートしています。

元のアプリケーションでは、このデータモデルを **Quote** クラスにカプセル化しています。Quote クラスを **TableView** のモデルとして利用するには、JavaFX プロパティを使用した JavaFX バインディングのサポートを追加する必要があります。テーブルに表示するそれぞれの列について、適切な型のプロパティ、そのプロパティのアクセッサ・メソッド、値の変更時に呼び出すメソッドが必要です。





ソッドは、株価の変動だけでなく、ポートフォリオの変更(株式の追加や削除など)も処理します。

あとは、`StocksMonitorMainWindow` クラスの中で `QuoteTable` を参照したり生成したりしている箇所を `QuoteFXTable` に書き換えれば、`TableView` の統合は完成です。

より面白く

ここまでで、JTable コンポーネントから TableView ノードへの置き換えが完了しました。悪くはないのですが、UIが大幅に向上するわけではありません。さらにJavaFX コードを追加して、もっと面白い画面を作成しましょう。

株価モニターとしては、値に変化があれば、減少は赤色、増加は緑色で強調したいところです。視覚的に変化をつけて

列のセル値ファクトリの設定に加えて、セル・ファクトリとして **Callback** オブジェクトを設定しています。このcallメソッドでは、**QuoteFXPriceCell** クラスの新しいインスタンスを返す処理だけを行っています。生成時に株式リストへの参照を渡しているのは、価格の変化を調べる必要があるからです。

セルの描画方法を変更するために、`updateItem` メソッドをオーバーライドします。正しく振る舞うようにするには、オーバーライド先のメソッドを最初に呼び出すことが重要です。`null` はセルの有効な値のため、`empty` フラグをチェックし、フラグが `true` の場合は何もしません。また、`null` 値もここでは意味がないため、値が `null` の場合もこのメソッドの処理を終了します。セルのテキストに、引数で渡された値を設定します。

**TableCell** は **Labeled** クラスを継承しています。**Labeled** クラスはテキストのラベルを持つすべてのコントロール(ボタンなど)に共通するクラスです。また、ラベルのテキストに加えグラフィカル・アイコンもサポートしています。このアイコンはノードであるため、必要に応じて完全なシーングラフとして扱うことができます。**QuoteCellGraphic** クラスは、色付きの三角形や四角形を使用して、価格の増加、減少、変化なしの状況を視覚的に示すノードを提供します。

コンストラクタで取得した株式リスト (quotes) の中を、**IndexedCell** インタフェースの **getIndex** を使用して参照できます。価格変化の状況を把握した後に、**QuoteCellGraphic** の該当するアイコンを表示できます。

価格を点減させるために **Timeline** を作成します。このクラスは、1 秒間にセルの透明度を1.0から0.1に変更し、また元に戻します。**cycleCount** を適切に設定することにより、セルを3回点減させます。

## 強力な機能

スタイルシートを編集すれば、コードを修正せずにアプリケーションの**ルック・アンド・フィール**を変えられます。

**スタイルシートは非常に強力**であり、多数のJavaFXノードがある場合でも、アプリケーションの外観をまったく別のものに 변경 することができ ます。

ユーザーの視線を引きつけるために、画面上で価格を点滅させることにしましょう。数値はフェードイン/フェードアウトさせてよりソフトな感じを出しましょう。

これらの実現には、価格列のセルを描画するためにカスタマイズした **TableCell** クラスを用意する必要があります。**リスト4**と**リスト5**に、**QuoteFXPriceCell** クラスのコードを示します。

リスト3の

QuoteFXTableNode クラスのコードをもう一度見てみると、価格

## リスト 4

## リスト 5

```
@Override
protected void updateItem(Float value, boolean empty) {
    super.updateItem(value, empty);

    if (empty)
        return;

    setText("" + value);
    icon = new QuoteCellGraphic();
    setGraphic(icon);

    float change = quotes.get(getIndex()).getChange().floatValue();

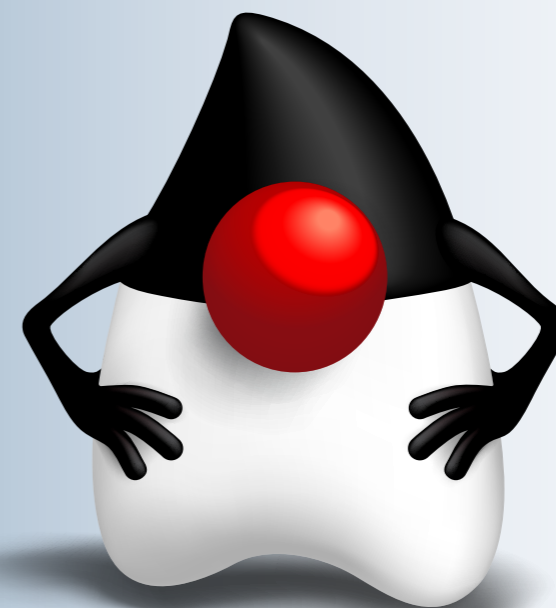
    if (change > 0.0)
        icon.setUp();
    else if (change < 0.0)
        icon.setDown();
    else icon.setNoChange();

    SimpleDoubleProperty faderProperty = new SimpleDoubleProperty();
    opacityProperty().bind(faderProperty);

    Timeline faderTimeline = new Timeline();
    faderTimeline.getKeyFrames().add(new KeyFrame(Duration.ZERO,
        new KeyValue(faderProperty, 1.0)));
    faderTimeline.getKeyFrames().add(new KeyFrame(new Duration(500),
        new KeyValue(faderProperty, 0.1)));
    faderTimeline.getKeyFrames().add(new KeyFrame(new Duration(1000),
        new KeyValue(faderProperty, 1.0)));
    faderTimeline.setCycleCount(3);
    faderTimeline.playFromStart();
}
```

 [全てのリストをテキストで表示](#)

- [JavaFX のドキュメントとチュートリアル](#)
- [JavaFX 2.0 API ドキュメント \(Javadoc\)](#)
- [JavaFX ショーケース](#)





ADAM BIEN



# 軽量パブリッシュ/サブスクライブ通信

Java EE 6とCDIを利用して無駄のない効率的なローカル・パブリッシュ/サブスクライブ通信を実装

インターネット時代の初期から存在するJ2EE仕様では、同期サービスを非同期に呼び出す方法、あるいはローカル・イベントを配信する方法としてJava Message Service (JMS)を回避策として利用してきました。Java EE 6は様々な開発容易性を取り入れた一方で、JMSに対する変更はなく、JMSは冗長的なAPIを提供しています。

Contexts and Dependency Injection (CDI) 1.0 (JSR-299) および Enterprise JavaBeans (EJB) 3.1 (JSR-318) を利用した場合、そのようなJ2EEの回避策やパターンが不要になります。JMSは本来のメッセージングにとつては極めて重要な技術ですが、非同期処理の実装やローカルのオブザーバ・パターン (Observer/パターン) の実装にはもうJMSは不要となります。

## 非同期呼出し

サービス・アクティベータ・パターン (Service Activator/パターン) の「背景 (Context)」に、次のような記述があります。「Enterprise Beanやその他のビジネス・サービスには非同期でアクティブ化するための手法が必要である」。サービス・アクティベータ・パターンはEJB

2.xコンポーネントを「正しい手続きで」非同期的に呼び出す唯一の手段を提供していました。サービス・アクティベータは、「既存のEJBインタフェースに対する横断的な関心事として非同期の性質を付加したデコレータ」と考えることができます。

このデコレーションは通常、`javax.jms.MessageProducer` によるJMSキュー (送信側) とメッセージ・ドリブンBean (非同期の受信側) で構成されます。しかしJava EE 6でサービス・アクティベータを実装しても、無駄な部分が生じます。`javax.jms.Queue` と `javax.jms.ConnectionFactory` を利用してトランザクション内でメッセージの送信側を実装し、`javax.jms.Message` インスタンスを作成して送信する必要があるからです。このような実装方法でJava EE 6が有用な点は、Dependency Injection (JSR-330) でリソースを取得できる点だけです。

**リスト1**に、NetBeans 7を使用して、`javax.jms.MessageProducer` を

EJB 3.1 beanでラップしたコード例を記述します (Microsoft Windowsでは[Alt+Insert]キー、Macでは[Ctrl+I]キー)。しかし、生成後のソース・コードは、プロジェクトのライフサイクル全体にわたって保守しなければなりません。

キューの反対側にあるメッセージ・ドリブンBeanはこのメッセージを受信し、それを `javax.jms.TextMessage` にキャストしてペイロードを抽出し、同期サービスに渡します。**リスト2**にこの非同期の受信側コードを示します。

**リスト2**のメッセージ・ドリブンBeanは、同期的なEJBを非同期に呼び出す目的でのみ使用します。JMSの機能や性質はまったく使用せず、アプリケーション・サーバーのプールから取得し

たスレッドで、ノン・ブロッキング呼出しを行うことにのみフォーカスしています。実際のビジネス・ロジックは**リスト3**のようになります。EJB 3.1のステートレス・セッションBeanである `SynchronousService` をメッセージ・ドリブンBeanである

`ServiceActivator` にインジェクトし、その後 `ServiceActivator` の非同期の `onMessage` メソッドで `SynchronousService` の `message(String message)` メソッドをデコレートしています。

**注:**ここで紹介した実装例は、すでにJava EE 6で書き直されシンプルになっています。元のJ2EE実装ではDependency Injectionを利用できません。そのため、EJB 2コンポーネントで他のコンポーネントにアクセスするためにJava Naming and Directory Interface (JNDI) でルックアップを行い、ローカルの `EJBHome` とローカル・インタフェースを実装し、さらに従来型の標準XMLデプロイメント・ディスクリプタを用意する必要があります。サービス・アクティベータは簡単な課題に対しても複雑な解決策が必要でした。

## Java EE 6の答え

Java EE 6の登場により、上記のようなサービス・アクティベータ・パターンの実装は時代遅れになりました。**リスト4**に、サービス・アクティベータと同等の処理をJava EE 6で記述したコード例

### シンプルなJMS

**JMS 2.0**では、シンプルな仕組みと、Java EE 7の各仕様、特にCDIとの綿密な統合を主な目標としています。



のクラスや関心のあるメッセージの受信に  
対してのみ注目します。

このパターンはネットワークのスケラビリティを向上し、より動的なネットワーク・トポロジを可能にします。

パブリッシュ/サブスクライブのおもなポイントはその動的な性質にあります。リスナーやブロードキャスタがその時々で異なることもあります。リスナーとブロードキャスタの間は動的なN:M関係があり、両方が存在する必要はありません。ブロードキャスタはリスナーが存在しなくてもメッセージを送信できます。ブロードキャスタがない状態でも複数のリスナーが待機できます。ここで、パブリッシュ/サブスクライブの定義上の「メッセージ」は

`javax.jms.Message`である必要はありません。メッセージを「イベント」や「ペイロード」と考えるとわかりやすいでしょう。CDIとEJB 3.1では、JMSを使用せずに無駄のないパブリッシュ/サブスクライブ通信を実装できます。

Java EE 6 (特に CDI) には、**javax.enterprise.event.Event** というインジェクション可能なイベントが付属しています。

これを利用して、任意のオブジェクトをローカルの全リスナーに対して配信 (fire) できます。リスナーがない場合、ブロードキャストされたペイロードは単に消失します。メッセージを送信するために必要な操作は、**Event** クラスをインジェクションし、配信したいペイロードを決めるだけです。**リスト6**に、CDIでイベントをインジェクトし、引数で指定した **String** プ

ロードキャストするコード例を示します。

ここでは **fire** メソッドを1回呼び出してメッセージを送信しています。CDIイベントはトランザクションを認識可能で、EJB 3.1のセッションBeanは通常トランザクションを利用可能です。この例では、何の設定(アノテーションやXML)もしていないため、broadcastメソッドはトランザクション内部で実行されます。

リスナーは、@Observesアノテーションを付加した1つのパラメータを持つ**void** メソッドを実装する必要があります。**リスト7**にCDIの **String** リスナーのコード例を示します。**@Observes** アノテーションが示すパラメータの型は、送信されるペイロードの型と一致する必要があります。その場合に限りメッセージが配信され、その他の場合メッセージは単に消失します。

JMSとは異なり、CDIを利用すると、成功したトランザクションと失敗したトランザクションを1つのリスナーを利用して一度に簡単に観察できます。必要な操作は `@Observes` アノテーションの `during` 要素の設定だけです。**リスト8**の例は、コミットとロールバックを一度にリスニングする方法を示しています。メッセージの配信は、指定したトランザクション・フェーズでのみ行います。特にバッチ・ジョブの監視や監査を実装する場合に、この `@Observes(during=)` 要素を利用することでCDIイベントの使い勝手がよくなります。これにより成功したトランザクションと失敗したトランザクションのすべてを簡単に追跡できます。

## 複数の宛先があるケース

CDIイベント機能は型安全です。送信側の型と監視側の型が一致すればイベントを配信し、それ以外の場合はイベントを無視します。このままのアプローチでは型を1回しか利用できず、重要なメッセージと重要でないメッセージを区別する方法がありません。も

リスト 6      リスト 7      リスト 8      リスト 9      リスト 10

```
@Stateless
public class MessageBroadcaster {
    @Inject
    Event<String> event;
    public void broadcast(String message){
        event.fire(message);
    }
}
```

 [全てのリストをテキストで表示](#)

もちろん、文字列をカスタム・イベント・クラスでラップしてメッセージの型を拡張するのも1つの策です。**リスト9**に、イベント・ディスパッチ用のカスタム・イベント・クラスの例を示します。

しかし、類似する宛先に対してそれぞれに合う型を適応するのは良いアプローチではありません。クラス数が増加し不要にパッケージが膨張してしまいます。これに対し、CDIでは修飾子を利用することで、この問題を明かに解決できます。**リスト10**に、メッセージ・ディスパッチ用の修飾子を示します。修飾子は、**@Qualifier** i ンタフェースで示すアノテーションです。**@Retention** には常に **RUNTIME** を設定し、**@Target** にはこのアノテーションの適用先の要素を指定する必要があります。**@Qualifier** アノテーションは型の拡張と考えることができます。CDIフレームワークはマッチング処理において、型だけではなく適用した修飾子も考慮します。この修飾子をインジェクション・ポイントにのみ利用します。

```
@Inject @Importance
(Importance.Degree.HIGH)
Event<String> event;
```

さらに、対応するパラメータに対してアノテーションを付加します。

```
public void onImportantMessage
(@Observes @Importance
(Importance.Degree.HIGH)
String message){}.
```

さらに、もう1つの作業が残っています。指定した修飾子を持つすべてのチャンネルに **Event** をインジェクトする必要があります。異なる修飾子を持つ **Event** を何回もインジェクトするのではなく、その場で **Qualifier** を選択できます。**リスト11**に、このような動的なチャンネル選択の方法を示します。メッセージ・リスナーをその場で選択する場合に必要なのは、**select** メソッドに **Qualifier** インスタンスを渡すだけです。ただし、アノテーションをインスタンス化できないという問題があるため、アノテーションのインスタンス化のためのヘルパー・クラスを作成します (**リスト12**)。

リスト12の **ImportanceSelector** クラスは **Importance** アノテーションを実装し、その列挙値とアノテーションの型を返します。また、コンストラクタのパラメータに列挙型の **Degree** を渡しています。これを利用して必要なメッセー





JULIEN PONGE



## Part 1

# GlassFishで実現するシンプルな クラスタリングと高可用性

GlassFishのクラスタリング機能で複数のサーバーから成るクラスタを運用し、アプリケーションの水平スケーラビリティを確保

**G**lassFishはJava EE標準仕様の参照実装として知られていますが、特に開発タスクに適し、高速モジュール化対応サーバーとしても有名です。もともと、GlassFishの可能性は開発フェーズにとどまりません。GlassFishは、Java EEの最新技術を使用して実装する必要のあるアプリケーションに適し、本番環境でも適用可能な軽量アプリケーション・サーバーでもあるのです。

**注:** GlassFishにはGlassFish Server Open Source EditionとOracle GlassFish Serverという2つのエディ

ションがあります。この記事はどちらのエディションにも適用可能な内容です。

本番環境では、アプリケーションに対する高可用性とトラフィックに対する負荷の軽減という2つの問題に対応する必要があります。その解決策は、1台のサーバーで運用するだけではなく複数のサーバーから構成するクラスタ環境を構築・運用し、アプリケーションの水平スケーラビリティを確保することで対応します。クラスタ内部で稼働する各サーバー・インスタンスがセッション・データをレプリ

ケーションできれば、一部のインスタンスがクラスタに参加するとき、あるいはクラスタから離脱するときにもアプリケーションが利用可能な状態を持続できるようになるため、高可用性を実現できます。

GlassFishのクラス  
タリング機能につい  
て、これより全2回のシ  
リーズで説明します。  
パート1のこの記事

では、GlassFishサーバーで構成するクラスタ環境の一元的なプロビジョニング方法と管理方法について紹介します。パート2では、GlassFishクラスタでアプリケーションの高可用性を実現する方法について解説します。

注:この記事で紹介するサンプルのソース・コードは[こちらからダウンロード](#)できます。

## 実行サンプルClockEEの紹介

この記事では、GlassFishクラスタの管理に的を絞るため、意図的に単純化した実行サンプルを利用します。現

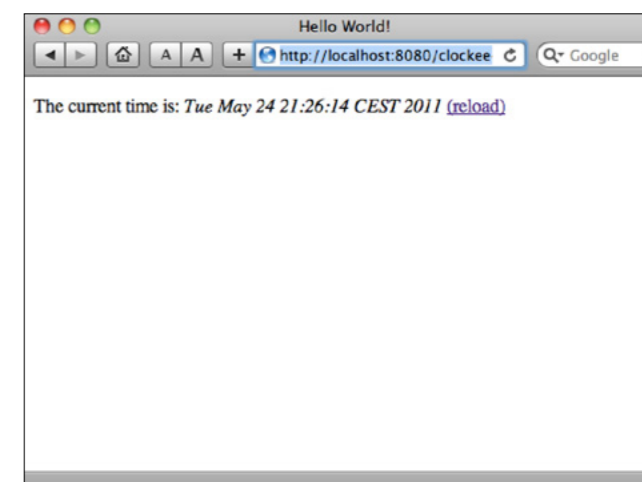


图1

在の時間を示す1枚のページを表示する、ClockEEという名前のWebアプリケーションです(図1)。

このアプリケーションはMavenプロジェクトであり、ソース・コードはClockEE.zipファイルに含まれています。唯一のビューを、JavaServer Facesを利用して表示します(リスト1)。

次にこのビューは、時刻の値を提供するContexts and Dependency Injection (CDI) 管理下のBeanを要求します(リスト2)。

@Named アノテーションは、この Bean のインスタンスを「clock」という名前で CDI から利用できるように



Julien Ponge provides a brief introduction to his GlassFish series.

PHOTOGRAPH BY  
MATT BOSTOCK/GETTY IMAGES



## 初めてのクラスタ

DASはGlassFishサーバーの複数のローカル・インスタンスやリモート・インスタンスを管理できるため、クラスタを効率的に実現できます。そのため、アプリ

`domain1`のDASに対して `my-first-cluster` を実行した後も、GlassFishのインスタンスが存在しないため、このままではほとんど役に立ちません。

もっともインスタンスの作成は非常に簡単です。ここでは、2つのローカル・インスタンスを **my-first-cluster** の一部として作成しましょう。つまり、DAS用のものと同じホストで稼働する2つのインスタンスを作成します (**リスト4**)。

もちろん、独自のポート番号を指定してもかまいませんが、ほとんどの場合でデフォルトのポート番号を利用して自動設定する事が可能です。インスタンスは **my-first-cluster** に含まれていますが、まだ起動されていないことを確認できます ([リスト5](#))。

また、クラスタに含まれるノードと、そのノードに属するインスタンスを確認できます(リスト7)。


## リスト 3

## リスト 4

## リスト 5

## リスト 6

## リスト 7

 [全てのリストをテキストで表示](#)

ではここで、クラスタにClockEEをデプロイしましょう。

## RESTについて

ケーションのデプロイ、設定、リソース管理を実行するコマンドを **asadmin** に発行して、1台のホストからクラスタ全体を一元的に管理することが可能です。発行したコマンドは自動的に各ホストにコピーされます。

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2012



图4

ノードについて  
SSHノードは、DAS  
からすべての管  
理を行うことが  
可能です。

SSHノードにはさらに興味深い機能がありますので、以降はその機能に絞って説明します。Configノードをクラスタに追加するには、ノード・ホストの準備が必要です。そのためには、まずGlassFishをインストールし、次に各ホストにローカルのConfigノードを作成し、DAS

に対してそのノードを宣言しなければなりません。次のセクションで示すとおり、SSHノードの場合はこれらのタスクを完全に自動化できます。

## リモートからのSSHノードのプロビジョニングと追加

このセクションでは、リモートからプロビジョニングし、GlassFishインスタンスをクラスタに追加する方法を確認します。前提として、リモート・サーバーの有効なユーザー・アカウントを持っており、そのサーバーの最小構成としてJava SE 6ランタイム環境とOpenSSHサーバーが稼働している必要があります。

OpenSSHが稼働していれば、テストの目的でローカル・マシン上にこの構成を用意してもかまいません。この記事の執筆時には仮想マシンを利用して検証を行いました。実際は、[Oracle VM VirtualBox](#)上に、Linux Ubuntu Maverickオペレーティング・システムをインストールし仮想環境を構築しました。

## リスト 9

## リスト 10

## リスト 11

## リスト 12

```
$ asadmin setup-ssh --sshuser
julien 192.168.56.101
Enter SSH password for julien@192.168.56.101>
Copied keyfile /Users/julien/.ssh/id_rsa.pub to julien@192.168.56.101
Successfully connected to julien@192.168.56.101 using keyfile
/Users/julien/.ssh/id_rsa
Command setup-ssh executed successfully.
```

 全てのリストをテキストで表示

(圖4)。

SSH経由で処理を実行するため、リモート・サーバーへ接続するたびに認証が求められます。デフォルトでは毎回パスワードを入力する必要がありますが、GlassFishにはこの煩わしさを解消する方法を提供しています。公開鍵/秘密鍵のペアをローカルで生成し、公開鍵のフィンガープリントをリモート・サーバーの`~/.ssh/authorized_keys`ファイルに追加することで、毎回パスワードを入力せずにSSH接続が可能になります。

この作業は手動でも行えますが、GlassFishで自動化することもできます(リスト9)。

では、リモート・サーバーにGlassFishがインストールされていないことを前提として、リモート・プロビジョニングを実行して、リモート・マシンの/home/julien/glassfishv3にGlassFishをインストールしましょう(**リスト10**)。

これで、ローカル・イメージがSSH経由

でコピーされ、稼働中のGlassFishアプリケーションのリモート・プロビジョニングが完了しました。この状態で、リモート・サーバーにSSHノードを作成できます。

```
$ asadmin create-node-ssh  
--nodehost 192.168.56.101  
--installdir  
/home/julien/glassfishv3 ubuntuvm  
Command create-node-ssh executed  
successfully.
```

このSSHノードにインスタンスを作成し、**vm1**という名前を付け、**my-first-cluster**に追加します(リスト11)。

新規に作成したインスタンスはデフォルトで停止状態です。このインスタンスをコマンドを実行し起動できます(リスト 12)。

ClockEEアプリケーションを **my-first-cluster** に対してデプロイすると、**vm1** インスタンスの起動時にClockEEを透過的

図5

**图7**

图6

**图8**

にデプロイしていることを確認できます  
(図5)。

## Web管理コンソール

これまでは、コマンドラインから **asadmin** を利用してすべての管理作業を行いましたが、前述のとおり、方法はこれだけではありません。REST Webサービス・インタフェースを利用して、GlassFishとサード・パーティ製の管理システムを容易に

統合できます。また、直観的なグラフィカル・インタフェースを表示するWeb管理コンソールも利用できます。どちらも **asadmin** シェル・コマンドと並ぶ管理インタフェースです

図6、7、8のスクリーンショットは、これまでコマンドラインで実行したいいくつかのタスク

について、Web管理コンソールで実行した例を示しています。

## まとめ

この記事では、GlassFishクラスタの一元的なクラスタ・プロビジョニングと管理方法について紹介しました。SSHサーバーと

Java SEランタイムが稼働している場合、新しいホストをリモートからインストール・構成・起動可能な点が特にGlassFishの強みです。簡単なステートレス・アプリケーションをクラスタにデプロイし、日常的に使用するasadmin管理コマンドで、クラスタの状態の確認やインスタンスの操作を行いました。

この全2回のシリーズの次回の記事では、透過的なセッション・フェイルオーバーによって、クラスタにデプロイしたステートフル・アプリケーションの高可用性を実現する方法について確認します。

**LEARN MORE**

- [GlassFish の製品ドキュメント](#)
- ["Clustering in GlassFish Version 3.1"](#)
- [GlassFish ビデオ \(YouTube\)](#)
- [Real World Java EE Night Hacks—  
Dissecting the Business Tier](#) by Adam Bien  
([press.adam-bien.com](http://press.adam-bien.com), 2011)
- [Beginning Java EE 6 with GlassFish 3](#) by  
Antonio Goncalves (Apress, 2010)
- [The Java EE 6 Tutorial: Basic Concepts](#)  
fourth edition by Eric Jendrock,  
Ian Evans, Devika Gollapudi, Kim Haase,  
and Chinmayee Srivathsa  
(Prentice Hall, 2010)

## 次回予告

次回の記事では、クラスター環境上でステートフルアプリケーションの高可用性を実現する方法について確認します。