1. はじめに *2*

2. JMS(Java Message Service) *2*

2.1. JMS のメッセージングモデル 3

3. OpenJMS *4*

3.1. インストール 4

3.2. OpenJMS の起動/停止 5

3.3. メッセージの送受信 6

3.4. 設定 8

3.4.1. ログ 8

3.4.2. メッセージの永続化 8

3.4.3. コネクタ 10

3.4.4. JNDI 設定 10

3.4.5. コネクションファクトリ設定 11

3.4.6. セキュリティ設定 11

3.4.7. キューとトピックの設定 12

3.4.8. ガベージコレクション 12

3.5. 設定ファイルリファレンス 12

3.6. 管理者用 API 13

4. まとめ *13*

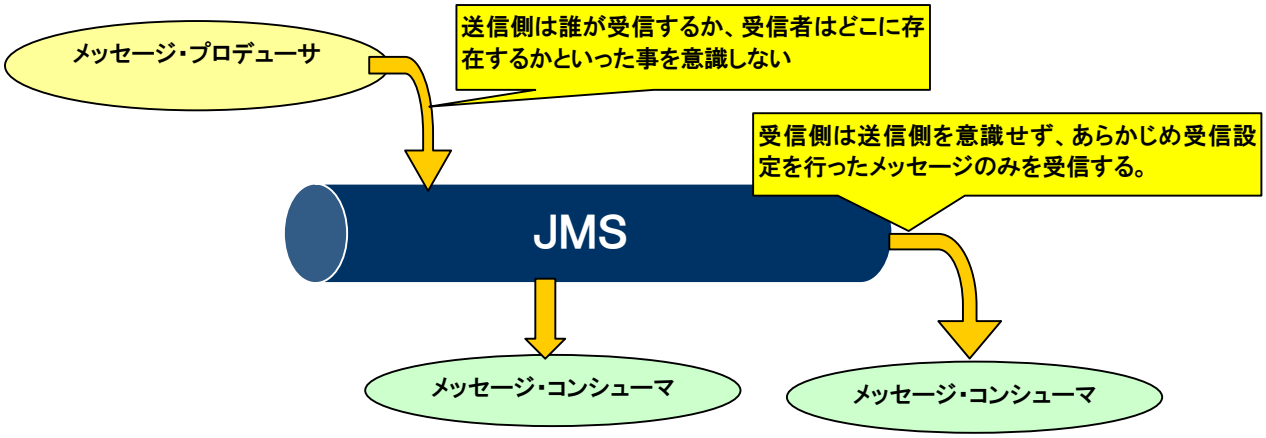
1. はじめに

本資料ではオープンソースの JMS 実装である OpenJMS(http://openjms.sourceforge.net)について説明します。まずは OpenJMS の説明に入る前に、JMS(Java Message Service)の基本的な考え方を見てみましょう。

2. JMS(Java Message Service)

JMS は J2EE1.3 から標準で含まれる「MOM」(メッセージ指向ミドルウェア)と呼ばれるメッセージングシステムにアクセスするための標準 API です。JMS でアプリケーション同士をつなぐと、各コンポーネントは自コンポーネントの処理する(送受信する)メッセージだけ意識すればよくなり、コンポーネント間の結合度を弱めることができるため、コンポーネント構成の柔軟な変更、耐障害性の向上といったメリットが得られます。

JMS では送信側をメッセージ・プロデューサ、受信側をメッセージ・コンシューマと



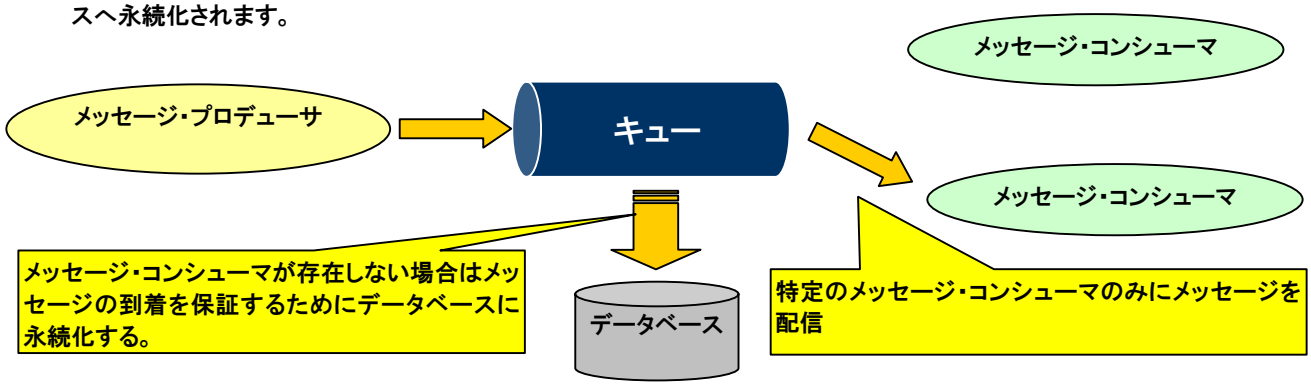
2.1. JMS のメッセージングモデル

JMS には以下の2種類のメッセーングモデルが存在します。

(1) PTP(Point To Point)

PTP メッセージングモデルは別の1つのコンポーネントに対してメッセージを送信するためのモデルです。キューに対してメッセージを送信し、受信されるとキューから削除されます。

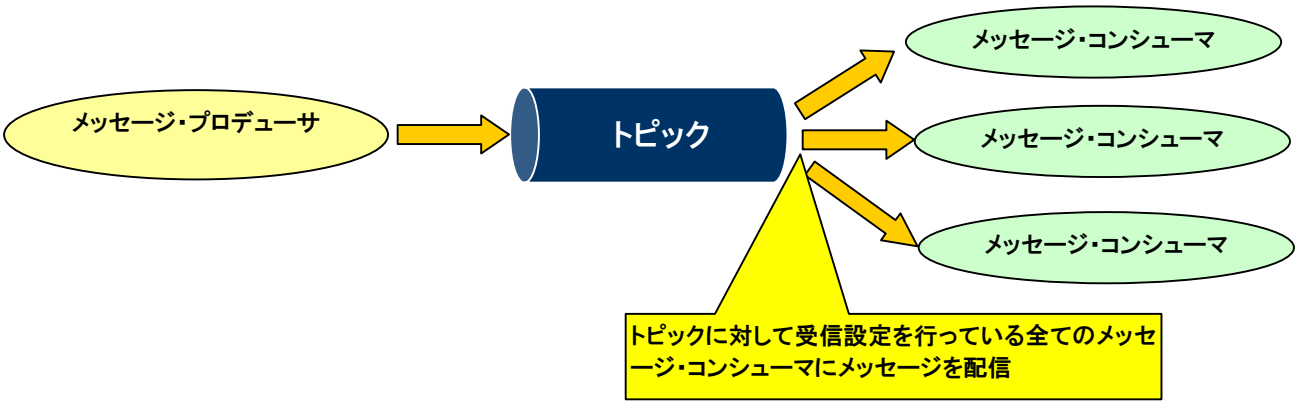
また PTP メッセージングモデルでは、メッセージの到着を保証するため、受信されなかったメッセージはデータベースへ永続化されます。



(2) Pub/Sub(Publisher/Subscriber)

Pub/Sub メッセージングモデルは、複数のアプリケーションに対してメッセージを送信するためのモデルで、トピックという単位で送受信を管理します。

送信側はトピックに対して送信し、受信側はトピックに対してリスナの設定を行っておきます。送信したメッセージは全てのリスナが受信した段階で削除されます。



3. OpenJMS

それでは、OpenJMS について見て行きましょう。

OpenJMS は JMS 1.0.2 に準拠したオープンソースの JMS 実装です(※)。オープンソースの JMS 実装にはこの他に JBoss(http://www.jboss.org/)に含まれる JBosMQ や ActiveMQ(http://activemq.codehaus.org/)などがありますが、OpenJMS の特徴としては非常にシンプルという事がいえます。

私も初めは JBossMQ の利用を考えていたのですが、JBossMQ ではオーバースペックで、もっと簡単に使いたいという思いから OpenJMS を使っています。

※JMS では API と参照実装のみ提供されており、各プロバイダが API に沿った形で実装しています。

それでは、さっそく OpenJMS をインストールして触れてみましょう。

なお、本資料では、OpenJMS のバージョン 0.7.6.1 を Windows 上で動かした場合をベースに説明します。Linux の場合は*.bat を*.sh と読み替えれば同じ手順で利用できます。

3.1. インストール

(a) OpenJMS のダウンロードページ(http://openjms.sourceforge.net/downloads.html)から最新バージョンをダウンロードします(今回は openjms-0.7.6.1.zip を使用)。

(b) 環境変数を設定します

環境変数	備考
JAVA_HOME	Java のインストールディレクトリ
OPENJMS_HOME	OpenJMS のインストールディレクトリ

以上でインストールは完了です。

3.2. OpenJMS の起動/停止

OpenJMS の起動/停止は管理ツールもしくはスクリプトから行います。

管理ツールは%OPENJMS_HOME%/bin/admin.bat を実行すると起動します。メニューの[Actions]から起動/停止、オンライン/オフラインの切り替えが出来ます。

図 3-1 OpenJMS の管理ツール

- スクリプトからの起動

  %OPENJMS_HOME%/bin/startup.bat を実行します。

- スクリプトからの停止

  %OPENJMS_HOME%/bin/shutdown.bat を実行します。

3.3. メッセージの送受信

さて、起動が出来たところでさっそくPTPおよびPub/Subのメッセージを送受信してみましょう。

メッセージの送受信を行う場合はJNDI経由で取得したコネクションファクトリからコネクションを取得し、PTPであればキュー、Pub/Subであればトピックを利用してメッセージの送受信を行います。

(1) PTPメッセージの送受信

PTPメッセージの送信処理

```
Hashtable properties = new Hashtable();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.exolab.jms.jndi.InitialContextFactory");
properties.put(Context.PROVIDER_URL, "rmi://localhost:1099/");

// Contextの取得
Context context = new InitialContext(properties);

// PTP用コネクションファクトリの取得
QueueConnectionFactory factory = (QueueConnectionFactory)context.lookup("JmsQueueConnectionFactory");
QueueConnection conn = factory.createQueueConnection();
conn.start();

// セッションの作成
QueueSession session = conn.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
// キューの取得
Queue queue = (Queue)context.lookup("queue1");

// メッセージの送信
QueueSender sender = session.createSender(queue);
TextMessage message = session.createTextMessage("HelloWorld");
sender.send(message);

// 終了処理
conn.stop();
session.close();
conn.close();
```

PTPメッセージの受信処理

```
Hashtable properties = new Hashtable();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.exolab.jms.jndi.InitialContextFactory");
properties.put(Context.PROVIDER_URL, "rmi://localhost:1099/");

// Contextの取得
Context context = new InitialContext(properties);

// PTP用コネクションファクトリの取得
QueueConnectionFactory factory = (QueueConnectionFactory)context.lookup("JmsQueueConnectionFactory");
QueueConnection conn = factory.createQueueConnection();
conn.start();

// セッションの作成
QueueSession session = conn.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
// キューの取得
Queue queue = (Queue)context.lookup("queue1");

// メッセージの受信
QueueReceiver receiver = session.createReceiver(queue);
TextMessage message = (TextMessage)receiver.receive();
System.out.println("Received message: " + message.getText());

// 終了処理
conn.stop();
session.close();
conn.close();
```

(2) Pub/Sub メッセージの送受信

Pub/Sub メッセージの送信処理

```
Hashtable properties = new Hashtable();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.exolab.jms.jndi.InitialContextFactory");
properties.put(Context.PROVIDER_URL, "rmi://localhost:1099/");

// Context の取得
Context context = new InitialContext(properties);

// Pub/Sub 用コネクションファクトリの取得
TopicConnectionFactory factory = (TopicConnectionFactory)context.lookup("JmsTopicConnectionFactory");
TopicConnection conn = factory.createTopicConnection();
conn.start();

// セッションの作成
TopicSession session = conn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
// トピックの取得
Topic topic = (Topic)context.lookup("topic1");

// メッセージの送信
TopicPublisher publisher = session.createPublisher(topic_);
TextMessage message = session.createTextMessage(text);
publisher.publish(message);

// 終了処理
conn.stop();
session.close();
conn.close();
```

Pub/Sub メッセージの受信処理

```
Hashtable properties = new Hashtable();
properties.put(Context.INITIAL_CONTEXT_FACTORY, "org.exolab.jms.jndi.InitialContextFactory");
properties.put(Context.PROVIDER_URL, "rmi://localhost:1099/");

// Context の取得
Context context = new InitialContext(properties);

// Pub/Sub 用コネクションファクトリの取得
TopicConnectionFactory factory = (TopicConnectionFactory)context.lookup("JmsTopicConnectionFactory");
TopicConnection conn = factory.createTopicConnection();
conn.start();

// セッションの作成
TopicSession session = conn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
// トピックの取得
Topic topic = (Topic)context.lookup("topic1");

// メッセージの受信
TopicSubscriber subscriber = session.createSubscriber(topic_);
TextMessage message = (TextMessage)subscriber.receive();
System.out.println("Received message: " + message.getText());


// 終了処理
conn.stop();
session.close();
conn.close();
```

3.4. 設定

3.4.1. ログ

OpenJMS のログを出力するためには、openjms.xml に以下の様に LoggerConfiguration 定義で、log4j の XML 形式の設定ファイルを指定します。

```
<LoggerConfiguration file="${openjms.home}¥config¥log4j.xml" />
```

3.4.2. メッセージの永続化

メッセージの永続化を使用するためには、メッセージ情報を保存するためのデータベース設定を行う必要があります。

なお、テスト済みの永続化可能なデータベースは以下となっていますが、今回は敢えて PostgreSQL を使用して確認しています。

データベース	バージョン
Oracle8i	8.1.7
Sybase ASE	12.0
MySQL	3.23.39
HSQLDB	1.6.1

永続化の設定手順は以下になります。

(c) JDBC ドライバをクラスパスに設定する

%OPENJMS_HOME%/bin/setenv.bat を開き、ファイル中の以下の行を変更する。

```
rem     set CLASSPATH=<insert path to JDBC driver here>
```

(d) 設定ファイルを変更する

%OPENJMS_HOME%/config/openjms.xml を開き、環境に合わせて設定を追加する。こちらは通常通りの JDBC 設定なので悩むことは無いでしょう。

```
<DatabaseConfiguration>
  <RdbmsDatabaseConfiguration
    driver="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:oci8:@myhost"
    user="openjms"
    password="openjms" />
</DatabaseConfiguration>
```

(e) データベースに永続化用のテーブルを生成する

次に、メッセージを保存するためのテーブルを作成します。以下のコマンドで上手くいかない場合は%OPENJMS_HOME%/config/db 配下の*.sql からし要するデータベースに合わせて実行します。

```
cd %OPENJMS_HOME%¥bin
dbtool.bat -create -config %OPENJMS_HOME%¥config¥openjms.xml
```

以上で永続化の設定は完了ですが、パフォーマンスを考えるとコネクションプールが必要になります。コネクションプールを使用する場合は以下の赤字の様に設定を追加します。

```
<DatabaseConfiguration>
  <RdbmsDatabaseConfiguration
    driver="org.gjt.mm.mysql.Driver"
    url="jdbc:mysql://localhost/openjms"
    user="openjms"
    password="openjms"
    maxActive="10"
    maxIdle="5"
    evictionInterval="3600"
    testQuery="select current_date"/>
</DatabaseConfiguration>
```

3.4.3. コネクタ

OpenJMS では、RMI、TCP、HTTP、SSL による接続が可能です。利用する方式によってパフォーマンスにも差が出てきますので、適材適所でプロトコルを選択します。

デフォルトでは以下の様に RMI コネクタを使用したキューとトピックのファクトリが定義されています。

```
<Connectors>
  <Connector scheme="rmi">
    <ConnectionFactories>
      <QueueConnectionFactory name="QueueConnectionFactory"/>
      <TopicConnectionFactory name="TopicConnectionFactory"/>
    </ConnectionFactories>
  </Connector>
</Connectors>
```

RMI の他に以下のコネクタが利用可能です。

コネクタ	備考
RMI	通信に RMI を使用するコネクタ
TCP	通信に TCP/IP を使用するコネクタ
TCPS	
HTTP	通信に HTTP を使用するコネクタ
HTTPS	通信に HTTPS を使用するコネクタ
Embedded	同一 VM 上のコネクタ

なお、複数のコネクタを同時に使用する場合はコネクタの名称を重複しないように設定する必要があります。

3.4.4. JNDI 設定

OpenJMS ではクライアントが、JNDI を利用してコネクションファクトリを取得します。

デフォルトでは以下の様に組み込み済みの JNDI プロバイダを利用する設定になっています。

```
<ServerConfiguration host="localhost" embeddedJNDI="true" />
```

また、外部の JNDI プロバイダを使用する場合は、以下の様に embeddedJNDI に false を設定し、JndiConfiguration を指定します。

```
<ServerConfiguration embeddedJNDI="false" />

<JndiConfiguration>
  <property name="java.naming.factory.initial"
            value="com.sun.jndi.rmi.registry.RegistryContextFactory" />
  <property name="java.naming.provider.url"
            value="rmi://localhost:1099" />
</JndiConfiguration>
```

3.4.5. コネクションファクトリ設定

コネクションファクトリでは、メッセージを送受信するためのキューやトピックの生成を行います。作成されたコネクションファクトリをクライアントが JNDI 経由で取得し、メッセージの送受信を行います。

コネクションファクトリには以下の 4 種類が用意されていますので、必要に応じて定義を追加します。

コネクションファクトリ	必須	備考
QueueConnectionFactory	YES	PTP 用コネクションファクトリ。
TopicConnectionFactory	YES	Pub/Sub 用コネクションファクトリ。
XAQueueConnectionFactory	NO	PTP 用コネクションファクトリ。XA トランザクションをサポートします。
XATopicConnectionFactory	NO	Pub/Sub 用コネクションファクトリ。XA トランザクションをサポートします。

3.4.6. セキュリティ設定

OpenJMS ではコネクションに対してセキュリティ設定を行うことが出来ます。セキュリティ設定を有効にするには securityEnabled を true にしてユーザ定義を追加します。

```
<SecurityConfiguration securityEnabled="true"/>

<Users>
  <User name="admin" password="openjms"/>
  <User name="user1" password="password1"/>
  <User name="user2" password="password2"/>
</Users>
```

3.4.7. キューとトピックの設定

ここでは、キューとトピックの設定について説明します。

使用するキューとトピックはあらかじめ設定ファイルに定義しておく必要があり、クライアントから JNDI で取得する場合はここで指定した"name"と一致する必要があります。

以下の例(デフォルトの状態)では、topic1 という Pub/Sub 用のトピックと、queue1,queue2,queue3 という PTP 用の 3 つのキューが定義されています。

なお、キューとトピックの設定は、管理者用 GUI や、管理用 API からも行うことが出来ます。

```
<AdministeredDestinations>
  <AdministeredTopic name="topic1">
    <Subscriber name="sub1" />
    <Subscriber name="sub2" />
  </AdministeredTopic>

  <AdministeredQueue name="queue1" />
  <AdministeredQueue name="queue2" />
  <AdministeredQueue name="queue3" />
</AdministeredDestinations>
```

3.4.8. ガベージコレクション

OpenJMS のメモリ管理関連の設定について説明します。

ガベージコレクションの対象となるのは、JMSExpiration で設定された期限切れのメッセージと、キャッシュされたメッセージです。また一時的にキャッシュされた永続化メッセージもガベージコレクションの対象になります。

```
<GarbageCollectionConfiguration memoryCheckInterval="30"
                                lowWaterThreshold="20"
                                garbageCollectionInterval="600"
                                garbageCollectionThreadPriority="5"/>
```

memoryCheckInterval・・・メモリの使用状況をチェックする間隔(秒)

lowWaterThreshold・・・トータルメモリに占める空メモリの割合(%)。空きメモリがここで指定した割合以下になるとガベージコレクションが実行される

garbageCollectionInterval・・・ガベージコレクションを実行する間隔(秒)

garbageCollectionThreadPriority・・・ガベージコレクションのスレッドの優先度(1～10)

3.5. 設定ファイルリファレンス

ここで紹介した以外の設定情報は設定リファレンス(http://openjms.sourceforge.net/config/reference.html)を参照してください。

3.6. 管理用 API

次に、管理用に用意されている API の利用方法について説明します。

管理用 API を利用する事で、プログラムから現在の構成の更新や表示を行うことが出来ます。

```
// 管理用コネクションファクトリの作成
String url = "rmi://localhost:1099/";
JmsAdminServerIfc admin = AdminConnectionFactory.create(url);

// 現在構成されているキュー&トピックの表示
Vector destinations = admin.getAllDestinations();
Iterator iterator = destinations.iterator();
while (iterator.hasNext()) {
   Destination destination = (Destination) iterator.next();
   if (destination instanceof Queue) {
       Queue queue = (Queue) destination;
       System.out.println("queue:" + queue.getQueueName());
   } else {
       Topic topic = (Topic) destination;
       System.out.println("topic:" + topic.getTopicName());
   }
}

// queue1 のメッセージ数の表示
String queue = "queue1";
int count = admin.getQueueMessageCount(queue);
System.out.println("Queue " + queue + " has " + count + " messages");

// topic1 のメッセージ数の表示
String topic = "topic1";
String name = "sub1";
count = admin.getDurableConsumerMessageCount(topic, name);
System.out.println("Subscriber " + name + " has " + count + " messages "
                        + "for topic " + topic);

// コネクションのクローズ
admin.close();
```

4. まとめ

OpenJMS について見てきましたがいかがでしたでしょうか。

機会があれば、その他のオープンソースの JMS 実装との比較などについても書いて行こうと思います。

記述内容に何かお気づきの点、質問等ありましたら下記までご連絡ください。

開発部
横井　朗
yokoi@bbreak.co.jp