ＰＳＬＸプラットフォーム計画

Java 版

共通コンポーネント実装マニュアル

第 3 部

PPS メッセージングサービス

バージョン 1.0

2009 年 6 月

NPO 法人ものづくり APS 推進機構

改訂履歴

日付	内容	備考
2009/03/13	バージョン 1.0 ベータ版	
2009/06/04	バージョン 1.0	

もくじ

1. はじめに

◆ 目的

PPS 共通コンポーネントは、PSLX プラットフォーム対応のソフトウェア構築において、異なるアプリケーションプログラムおよびソフトウェア環境で相互接続性を保証するための共通の実装環境です。PPS 共通コンポーネントは、OASIS PPS 技術委員会が定めた国際標準に沿って実装されています。またメッセージ通信に関する詳細な知識および XML に関する詳細な知識を必要とせずに容易に PSLX プラットフォーム対応のシステム構築が可能となるように設計されています。

本マニュアルは、実装マニュアル第 3 部「PPS メッセージングサービス Java 版」です。本マニュアルは、Java における PSLX プラットフォームによる業務アプリケーション間のメッセージの送受信に関するプログラミングを支援するためのコンポーネントの利用方法を解説します。メッセージキューを利用したメッセージの送受信や、Web サーバを利用した SOAP および HTTP／POST を利用した送受信を行うプログラムの実装方法について解説します。

◆ 対象とする読者

（1）資格

本マニュアルは、PSLX プラットフォーム計画プロジェクトに参加している企業の従業員に対して、PSLX プラットフォーム対応ソフトウェアを開発するために公開している文章です。PSLX プラットフォーム計画プロジェクトのメンバー以外であっても、本マニュアルを閲覧することは可能ですが、NPO 法人ものづくり APS 推進機構の許可なく複製や再配布を行うことは禁止されています。

（2）必要とする知識・技術

ソフトウェア開発の一般知識を有する人を対象にした文章です。特に、下記の項目についての知識が必要です。

■ Java 言語によるプログラム開発

- メッセージキューによるメッセージ通信
- Web サーバによる動的アプリケーションの開発

◆ 動作環境

本マニュアルに含まれる内容は、次の環境で利用することを前提としています。

必要ソフトウェア環境

区分	内容
オペレーティングシステム	Windows XP Service Pack 3 以降 Windows 2003 Server 以降
コンポーネント	JRE 5.0 以降および JDK5.0 以降 Apache Tomcat 5.0
Web ブラウザ	Microsoft Internet Explorer 6,7 および 8 Mozilla Firefox 3.0
開発ツール	Eclipse 3.2 などの Java 開発環境

（1） 実行環境

本マニュアルに記載されているすべてのプログラムを実行するためには、次のメッセージキューサーバのいずれか 1 つが、実行するコンピュータか、あるいは実行するコンピュータから LAN 経由でアクセス可能な他のコンピュータ上にインストールされている必要があります。メッセージキューサーバのインストール方法については、次の各項目に対応した URL を参照してください。

名称	開発元	URL
Apache ActiveMQ	Apache Software Foundation	http://activemq.apache.org/
WebSphere MQ	IBM	http://www.ibm.com/developerworks/jp/websphere/category/wmq/

2. 通信方式の基本概念

◆ さまざまな通信方式の対応

PPS メッセージングサービスは、業務アプリケーション間での情報のやりとりを実際に行うためにさまざまな通信手段と業務アプリケーションとをつなぐための共通コンポーネントです。業務アプリケーション間との通信手段としては、SOAP やメッセージキューなどが挙げられ、より単純な手段としては HTTP や SMTP や FTP などが挙げられます。

PPS メッセージングサービスを利用することで、下位の通信手段を意識することなく、業務アプリケーション間での情報のやりとりを行うアプリケーションプログラムの開発が可能となります。下位の通信手段が変更になった場合でも、プログラムの作り直しを避けることができます。

通信手段としてメッセージキューサーバを利用することで、業務アプリケーション間をピア・ツー・ピア方式で通信することが可能です。また通信手段として SOAP や HTTP/POST を利用する場合は、サーバ・クライアント方式で通信することが可能です。SOAP や HTTP/POST を利用するためには、ASP.NET(IIS)や Servlet/Axis(Tomcat)、PHP などのアプリケーションサーバを利用します。

◆ 送信と受信

PPS メッセージングサービスのための共通コンポーネントとして、“org.pslx.PpsMessaging.jar”が提供されています。PPS メッセージングサービスには、メッセージキューおよび SOAP、HTTP/POST によって、送信者から受信者へテキストデータの送受信を行うために必要なライブラリが用意されています。メッセージキューを利用する場合には、送信者は、メッセージキューサーバにあらかじめ割り当てられたキューの名称を指定し、そこにメッセージを送信します。一方、受信者は、キューの名称を指定し、そこにある最も古いメッセージから順に取り出します。この結果、メッセージキューサーバを経由して、送信者から受信者へメッセージが送信されることになります。なお、メッセージキューの特徴として、キュー読み出しは、書き込んだ順となります。また一度キューから読み出しがら、その時点でキューには該当データが存在しなくなります。

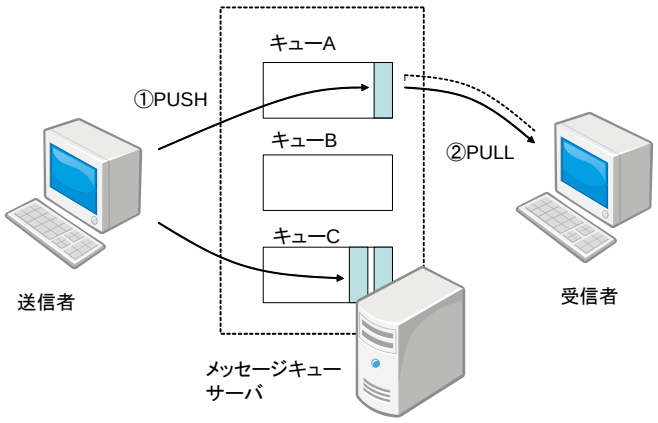


図 1　メッセージキューの基本機能

◆　トランザクションの構造

メッセージキューを用いた通信は、基本的には非同期で行われます。つまり、送信側が送信したタイミングと受信側が受信したタイミングはずれがあり、送信側は、受信側がメッセージを受信したかどうかにかかわらず次の処理を行うことが可能です。

一方で、業務情報の照会を行う場合や、受信確認をおこないたい場合など、受信者からの返信メッセージを、送信者へ返すようなトランザクションも存在します。このような場合には、次の図のように、メッセージキューサーバを介して、往信メッセージと返信メッセージが交換されます。

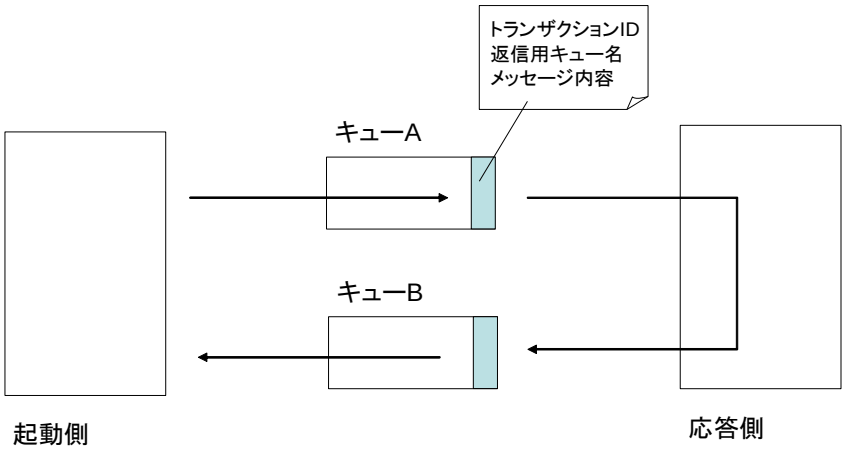


図 2 トランザクションの構造

ここで、起動側（クライアント）は、送信メッセージに、トランザクション ID と、自身

が管理している返信用キュー名を、送信内容と合わせて設定し、応答側が管理しているキューに送信します。

応答側（サーバ）は、受信したメッセージに返信が必要なものがある場合には、そこに設定している返信用キューに対して返信メッセージを送信します。この際に、返信メッセージのトランザクション ID は、受信メッセージのトランザクション ID と同一にします。

起動側は、当初のメッセージ送信後、返信用キューを監視し、そこに到着したメッセージを調べ、送信メッセージのトランザクション ID と同じ返信メッセージがあれば、その内容を送信メッセージと対応づけ、返信メッセージにある内容を処理します。

これらの一連の処理は、共通コンポーネントによって実施され、業務アプリケーション開発者は、それらをほとんど認識せずにプログラミングを行うことができます。

3. 必要なファイルとモジュール構成

◆ 必要なファイル

本マニュアルで解説する機能を実行するためには、次の実行ファイルおよび定義ファイルが必要となります。

（1）実行ファイル

ファイル名	説明
org.pslx.PpsMessaging.jar	PPS メッセージングサービスのパッケージ本体です。本マニュアルで取り上げるプログラムを実行するために必要なパッケージです。
org.pslx.PpsMessaging.Web.jar	SOAP クライアントおよび HTTP クライアントとして WWW サーバにアクセスするためのコンポーネントのパッケージです。
apache-activemq-4.1.2.jar または activemq-all-5.1.0.jar　など	(メッセージ通信に ActiveMQ を利用する場合のみ) Apache ActiveMQ および JMS を利用するためのパッケージです。これらは、ActiveMQ に含まれるものであり、バージョンによって名称が異なります。

（2）定義ファイル

ファイル名	説明
jndi.properties	JMS によるメッセージブローカへの接続先および使用するクラスを定義する設定ファイルです。

◆ モジュール構成

PPS メッセージングサービス(以下、本コンポーネント)を業務アプリケーションで利用する場合には、利用するメッセージキューサーバの種類によって、次のようにモジュール構成が異なります。どのメッセージキューサーバを利用するかは、定義ファイルによって指定します。

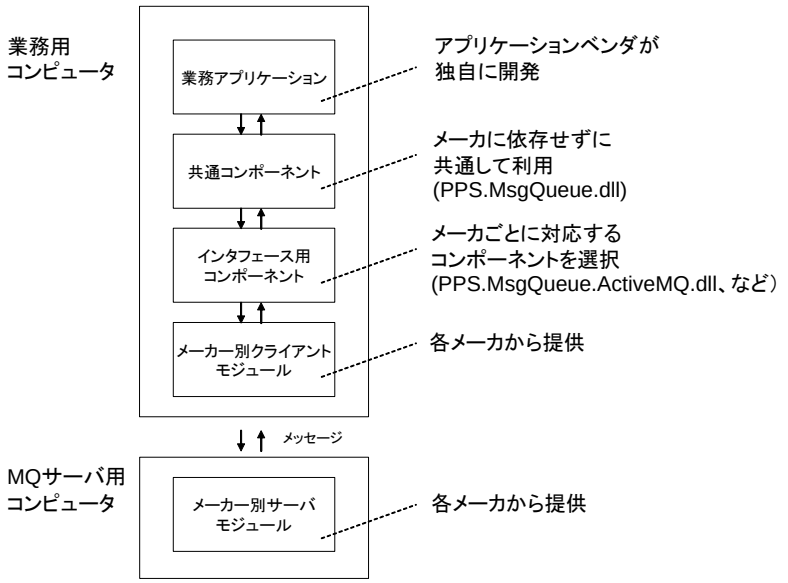


図 3 本コンポーネントを利用した業務アプリケーションの構成

◆ 定義ファイルの設定

本コンポーネントで通信手段として使用するメッセージキューサーバは、定義ファイル“jndi.properties”で設定します。この定義ファイルは、JNDI(Java Naming and Directory Interface)を使用したものであり、使用するメッセージキューサーバに応じて JNDI プロバイダおよび、メッセージキューサーバがあるホスト名およびポート番号を指定してください。また、メッセージキューで使用するキューは、この定義ファイルで設定します。本コンポーネントでは、キューの名称として JNDI 名を指定します。JNDI 名に対応する実際のキュー名は、この定義ファイルで設定してください。次の例は、メッセージキューサーバとして ActiveMQ を使用する場合の jndi.properties の設定例です。

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory

java.naming.provider.url = tcp://localhost:61616

useEmbeddedBroker = false

connectionFactoryNames = connectionFactory, queueConnectionFactory,
                                                    topicConnectionFactory
queue.TestQueue = TestQueue
queue.RequestQueue = RequestQueue
queue.ResponseQueue = ResponseQueue
```


4. メッセージの送信と受信

◆ 準備作業

本コンポーネントの API が含まれるパッケージは、org.plsx.Messaging.jar によって提供されています。業務アプリケーションで本コンポーネントを使用する場合には、クラスパスに org.plsx.Messaging.jar を追加する必要があります。

Java の開発環境として Eclipse を使用する場合は、プロジェクト項目を選択し、コンテキストメニューから「ビルド・パス」の「ビルド・パスの構成」を選び、「Java のビルド・パス」画面を表示します。「ライブラリー」タブから「外部 JAR の追加」で、本コンポーネントのパッケージを選択し、ビルドパスに追加します。また使用するメッセージキューサーバに応じて、別途パッケージをビルドパスに追加する必要があります。

Java プログラム内で本コンポーネントのクラスを利用する場合は、import 文でプログラムの先頭で本コンポーネントのパッケージ(org.plsx.Messaging)をインポートします。

```
import org.pslx.Messaging.*;
```

メッセージの送受信には、本コンポーネントに含まれるメッセージマネージャクラス（QueueManager）を使います。プログラムの初期化の時点で、このクラスのインスタンスを作成してください。このメッセージマネージャのインスタンス変数は、メッセージの送受信の際に常に利用するため、開発者が作成するクラスのフィールドで宣言することをおすすめします。

```
// PPSメッセージマネージャの定義
QueueManager messageManager = new QueueManager();
```

メッセージマネージャクラスのコンストラクタが実行される際に、前の章で解説した“jndi.properties”の設定に基づいて、メッセージキューサーバへコネクションクラスが生成されます。

◆ メッセージの送信

メッセージを送信するためには、send メソッドを使います。次のように、このメソッドの引数に、送信するテキストおよび送信先キューの名称を指定してください。送信先に指定したキューの名称は、JNDI によって名前解決され“jndi.properties”で指定した名称が使用されます。次のプログラムは、TestQueue に対して「こんにちは」というテキストメッセージを送信するプログラム例です。

```
QueueManager messageManager = new QueueManager();
try {
        // メッセージを送信します
        messageManager.send("こんにちは", "TestQueue");
} catch (Exception e) {
        e.printStackTrace();
}
```

◆ メッセージの受信

メッセージキューサーバからメッセージを受信するには、receiveMessage メソッドを使います。このメソッドは、メッセージキューサーバの指定したキューにメッセージが存在する場合、そのキューからメッセージを一つ受信します。メッセージを受信した時点で、そのキューから、受信したメッセージが取り除かれます。キューにメッセージが存在しない場合は、メソッドの戻り値として null を返します。

```
QueueManager messageManager = new QueueManager();

// 受信キューからメッセージを受け取ります
String receivedMessage = messageManager.receiveMessage("TestQueue");
if(receivedMessage != null) {
        System.out.println(receivedMessage);
} else {
        System.out.println("受信されたメッセージは、ありません。");
}
```


◆ 非同期のメッセージ受信

メッセージを受信するもう一つの方法として、指定したキューにメッセージが到着した際に、メッセージを受信する方法があります。この方法を非同期のメッセージ受信と呼びます。例えばメッセージを受け取ったときに処理を行うプログラムを作成する場合に、receiveMessage メソッドを使ったメッセージの受信では、メッセージが到着したかどうかを確認するために、頻繁に receiveMessage メソッドを実行する必要があります。一方、非同期のメッセージ受信では、メッセージを受信したことを受け付けるイベントリスナを一度設定しておき、メッセージキューサーバがメッセージを受け取った際に、イベントリスナで受け取ることができます。

非同期のメッセージ受信を行うには、まず次のように、指定したキューにメッセージが到着したことを通知するためのキューレシーバクラスのインスタンスを生成します。本コンポーネントの非同期のメッセージ受信は、JMS のクラスに依存します。メッセージマネージャでは、createQueueSession メソッドによって JMS のキューセッション(QueueSession クラスのインスタンス)を生成することができます。また getQueue メソッドによって指定したキューの名称である Queue クラスのインスタンスを取得できます。ここで指定したキューがメッセージの到着を監視するキューとなります。

その後、createQueueSession メソッドで生成したキューセッションを使って、キューレシーバ(QueueReceiver) キューセッションを生成し、setMessageListener メソッドによってイベントリスナを設定します。ここで指定したイベントリスナに対して、キューにメッセージが到着したことを通知します。

次の例では、MessageListener インスタンスを実装した SimpleReceiverAsync クラスに対してメッセージが到着したことを通知します。

```
QueueManager messageManager = new QueueManager();


//セッションの作成する
QueueSession session = messageManager.createQueueSession();
Queue queue = messageManager.getQueue("TestQueue");

//キューレシーバを生成する
QueueReceiver receiver = session.createReceiver(queue);

//メッセージキューの受付を開始する
receiver.setMessageListener(new SimpleReceiverAsync());
```



```
        messageManager.connect();

        //終了待機
        try{
                System.out.println("終了するには何かキーを押してください。");
                System.in.read();
        } catch(Exception e) {
                e.printStackTrace();
        }

        //受信に使用したレシーバ、セッション、接続を閉じます
        receiver.close();
        session.close();
        messageManager.disconnect();
```

次のプログラムにある onMessage メソッドは、JMS に含まれるイベントリスナ(MessageListener インタフェイス)に含まれるメソッドです。onMessage メソッドここで、引数(message)が、受信したメッセージになります。受信したメッセージの内容を取得するには、message の getText メソッドを使います。なお Message は、JMS の Message クラスのインスタンスです。

```
public class SimpleReceiverAsync implements MessageListener {

        /** キューからメッセージを受信すると実行される */
        public void onMessage(Message message) {
                try{
                        TextMessage textMessage = (TextMessage)message;

                        String xml = textMessage.getText();
                        System.out.println("メッセージを受け取りました。");
                        System.out.println(xml);

                } catch(Exception e) {
                        e.printStackTrace();
                }
        }
}
```

非同期メッセージの受信を開始するには、メッセージマネージャの connect メソッドを実行します。このメソッドが実行されるまでメッセージ到着の監視が開始されません。

また、プログラムの終了時などメッセージ到着の監視を終了する場合は、必ずキューレシーバやキューセッションの close メソッドをそれぞれ実行し、メッセージマネージャの d

isconnect メソッドを実行してください。メッセージキューサーバとの接続が確立している状態かどうかを取得するには、メッセージマネージャの isConnected メソッドを使います。プログラム終了時に、このメソッドの戻り値が true の場合は、次のようにキューレシーバやキューセッションの close メソッドでメッセージ到着の監視を終了させ、メッセージキューサーバとの接続を切断してください。

```
// 終了時に受信キューを閉じる
if (messageManager.isConnected()) {
    //受信に使用したレシーバ、セッション、接続を閉じます
    receiver.close();
    session.close();
    messageManager.disconnect();
}
```


5. クライアントとサーバの連携

◆ クライアントの処理

単にメッセージの送信または受信を行うプログラムは、シンプルであるのに対して、クライアントとサーバのプログラムは、多少複雑になります。この章では、シンプルなメッセージの送信および受信の組み合わせによって、クライアントの依頼メッセージに対してサーバが返信メッセージを送るプログラム例について解説します。

クライアントとサーバの連携の流れとして、まずクライアントでは、依頼メッセージをサーバが待ち受けるキューに送信します。このとき、依頼メッセージには、サーバからの返信メッセージを送る返信先のキューが含まれています。そして、クライアントは、その返信先のキューにサーバからの返信が到着するのを待ちます。返信先のキューにメッセージが到着した後、そのメッセージが依頼したメッセージに対する返信であるかどうかを確認して、もしそうであれば何らかの処理を行い終了します。

この一連の処理を行うには､本コンポーネントのメッセージマネージャの request メソッドを使います。request メソッドを使うことで、前に解説した一連の送受信処理を 1 つのメソッドで行うことが可能です。次のプログラムでは、request メソッドを使ったサーバへの依頼メッセージを送り、その返信メッセージを受信するプログラム例を示します。

```
String message = "送信するメッセージ";
String responseText;
try {
   // 依頼メッセージを送り、返信を受け取ります
   responseText = messageManager.request(message, "RequestQueue", "ResponseQueue");
} catch(PpsMessagingException e) {
   // タイムアウトの場合
   e.printStackTrace();
   responseText = null;
}

System.out.println("***** 戻り値 *****");
System.out.println(responseText);
```

上のプログラムにあるマネージャオブジェクトの request メソッドによって､サーバが監視しているキュー(RequestQueue)へ依頼メッセージを送信しています。最初の引数（mess

age）が依頼メッセージを含んだ文字列であり、2 つ目の引数がサーバの監視しているキューの名称、3 つ目の引数が要求メッセージに対する返信メッセージを送るキューの名称です。クライアントから送信した依頼メッセージに対するサーバの返信メッセージは、response メソッドの戻り値として返されます。このプログラムでは、responseText に返信メッセージが受信されます。

response メソッドによるメッセージ送信および受信は、同期処理として行われるため、このメソッドで依頼メッセージを送った後、サーバから返信メッセージを受け取るまで待ちます。そのため、返信メッセージを受け取るまでプログラムの制御は、戻りません。ただし、一定時間以上待っても返信がない場合には、タイムアウトの例外が発生します。

返信メッセージの受信のタイムアウトの時間は、setTimeOutLength メソッドで設定することができます。単位はミリ秒で、既定値は 3000(3 秒)に設定されています。また、setSleepingInterval メソッドでは、返信メッセージを受け取るまでの間、他のプロセスに制御を渡すための間隔を指定することができます。この値は、タイムアウト時間よりも短くしなければなりません。

```
messageManager.setTimeOutLength(3000);
messageManager.setSamplingInterval(100);
```

◆ 非同期クライアント

クライアント処理を非同期で行いたい場合には、まず返信メッセージを受信するキューへのメッセージの到着の監視を開始させた後に、依頼メッセージを送信します。返信メッセージを受け取った際には、受け取ったメッセージがクライアントが直後に送った依頼メッセージに対する返信メッセージであるのかどうかを確認します。この確認には、依頼メッセージのメッセージ ID(MessageID)および返信メッセージの相関 ID(CorrelationID)を使います。相関 ID とは、依頼メッセージとそれに対応する返信メッセージを対応づけるものであり、サーバは、依頼メッセージに設定されているメッセージ ID を、相関 ID として返信メッセージに設定してあります。なお、メッセージ ID および相関 ID は、JMS に依存しており、詳しくは JMS のドキュメントをご覧ください。

次のプログラムは、非同期受信を使ったクライアントのプログラム例です。このクライアントでは、サーバが依頼メッセージを待ち受けているキュー(RequestQueue)に対して依頼メッセージを送り、その返信メッセージを受け取るキュー(ResponseQueue)に返信メッ

セージが到着するのを待ちます。サーバに対して依頼メッセージを送り、応答メッセージを非同期で受信する場合は、requestAsync メソッドを使います。requestAsync メソッドの戻り値には、送信した要求メッセージが返されます。このメッセージに含まれるメッセージ ID を変数へ保存しておき、返信メッセージの受信時の確認に使います。

```
messageManager.connect();

//セッションの作成する
QueueSession session = messageManager.createQueueSession();
Queue queue = messageManager.getQueue("ResponseQueue");

//キューレシーバを生成する
QueueReceiver receiver = session.createReceiver(queue);

//依頼メッセージを送信する
System.out.println("依頼メッセージを送信しました。");
Message sentMessage = messageManager.requestAsync
                      ("依頼メッセージです。", "RequestQueue", "ResponseQueue");
```

サーバからの返信メッセージを受け取った際に実行される onMessage メソッドの内容は、次のようになります。ここで、message に含まれる getJMSCorrelationID メソッドによって相関 ID の照合が行われています。

```
//応答メッセージの受付処理
receiver.setMessageListener(new MessageListener() {
public void onMessage(Message message) {
    try{

        if(sentMessage.getJMSMessageID().equals(message.getJMSCorrelationID())) {
                TextMessage textMessage = (TextMessage)message;

                String text = textMessage.getText();
                System.out.println("メッセージを受け取りました。");
                System.out.println(text);
        } else {
                // その他の受信メッセージの処理を記述します。
                System.out.println("その他の受信メッセージを受け取りました。");
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
}
});
```


なお、プログラム終了時には、この返信メッセージの到着監視を終了するようにしてください。

```
//受信に使用したレシーバ、セッション、接続を閉じます
receiver.close();
session.close();
messageManager.disconnect();
```

◆ サーバの処理

サーバでは、クライアントからの依頼メッセージを処理し、その返信メッセージをクライアントが指定したキューに対して送信します。あらかじめクライアントが依頼メッセージを送る先の待ち受けキューの名称を公開しておきます。なおクライアントから送られた依頼メッセージを待ち受けるプログラムは、前の章で解説した「非同期のメッセージ受信」と同じです。

次のプログラムは、サーバ処理を行うプログラム例です。クライアントからの依頼メッセージを受け取るキュー(RequestQueue)の監視を監視します。

```
String requestQueueName = "RequestQueue";

//メッセージブローカへ接続します
messageManager.connect();

//セッションを生成します
receiveSession = messageManager.createQueueSession();
Queue receiveQueue = messageManager.getQueue(requestQueueName);

//レシーバを生成します
receiver = receiveSession.createReceiver(receiveQueue);

//メッセージキューの受付を開始します
receiver.setMessageListener(this);
```

待ち受けキューに依頼メッセージが到着した場合は、依頼メッセージを解析し、返信メッセージを送信します。依頼メッセージに対する返信メッセージを送る場合は、メッセー

ジマネージャの sendResponseOf メソッドを使います。このメソッドの 1 番目の引数に、作成する返信メッセージに対応する依頼メッセージを指定し、2 番目の引数に、返信先のキューの名称を指定します。このメソッドでは、依頼メッセージのメッセージ ID に対する相関 ID を設定した上で、返信メッセージを送信します。次のプログラムは、依頼メッセージを受信した後に、返信メッセージを送るプログラム例です。

```
receiver.setMessageListener(new MessageListener() {
/** キューからメッセージを受信すると実行されるメソッド */
public void onMessage(Message message) {
        try {
                //要求メッセージの取得
                TextMessage requestMessage = (TextMessage)message;
                String requestText = requestMessage.getText();
                System.out.println("メッセージを受け取りました");

                //応答メッセージの送信
                String responseText = "応答メッセージです。¥n" + requestText;
                messageManager.sendResponseOf(requestMessage, responseText);
                System.out.println("メッセージを送りました");
        } catch(Exception e) {
                e.printStackTrace();
        }
}
});
```

このように、本コンポーネントを利用して、比較的短いプログラムでクライアントおよびサーバを実装することができ、また、メッセージキューを使ったメッセージ通信を行うことが可能になります。

6. Web サーバの構築

◆ SOAP サーバの構築

PPS メッセージサービスを利用して、SOAP 形式でメッセージ通信を行う場合のサーバ構築方法について説明します。ここでは、Eclipse などの開発環境から Web サービスを作成し、要求メッセージに対する応答メッセージを返すサーバの構築例を挙げます。クライアントは、PPS メッセージサービスを通じて、ここで構築したサーバに対して SOAP 形式でメッセージを送受信します。

SOAP サーバのひな形プログラムは、PpsMessagingSoapServiceJava フォルダにあります。ひな形プログラムでは、org.pslx.webservice パッケージに含まれる PpsMessaging.java で次のように Web サービスとして利用できる request メソッドを定義していします。

```
public class PpsMessaging {

    public PpsMessaging() {
    }

    public String request(String xml, String senderId, String sendTo) {
        return "サーバ(SOAP経由)からの応答メッセージです。¥r¥n"
            + senderId + "から" + sendTo + "へ次のメッセージを受け取りました¥r¥n"
            + xml;
    }
}
```

request メソッドは、クライアントから PPS メッセージングサービスを使って送られたすべてのリクエストを処理します。引数 xml には、クライアントからの依頼メッセージが代入されています。引数 senderId には、依頼メッセージを送ったクライアントを区別する名称が代入されます。引数 sendTo は、依頼メッセージを送る先を表すコネクション名が代入されています。

サーバは、受け取ったリクエストにしたがって、受け取った XML の内容を解釈し、PPS の規約にしたがって、返信メッセージを生成し、戻り値として文字列を返してください。返信メッセージが必要ない場合には、長さ 0 の文字列を設定してください。

この Web サービスを Tomcat などのアプリケーションサーバへデプロイし、Web サーバ

上の特定のフォルダに設定され外部からアクセス可能な状態にします。クライアントには、その際のアドレスを事前に設定しておきます。

クライアントからこの Web サービスにアクセスするためのサンプルは、PpsMessaging WebJava フォルダに用意されています。

SOAP サーバとの間のメッセージ交換では、情報が常にクライアントからサーバに対するリクエストとそのレスポンスという形式となります。SOAP サーバ側から相手にリクエストを送ることはありません。通信相手が SOAP サーバとなるクライアントの場合には、PPS 共通コンポーネントを利用したプログラムには次の制約があります。

表 1 SOAP サーバとの通信における制約

区分	対象メソッド	対応内容
送信	sendMessage sendTextMessage	対応しています。
受信	receiveMessage receiveTextMessage	対応していません。サーバは、必ず要求メッセージを送信する必要があります。
送受信（同期）	requestMessage requestTextMessage	対応しています。メッセージキューによるサーバと同様に利用することが可能です。
送受信 （非同期１）	sendMessage receiveMessage	対応していません。
送受信 （非同期２）	sendMessage MessageConnector.receiveMessage	対応していません。

◆ WSDL の定義

PPS メッセージサービスを使った SOAP 形式によるメッセージ通信では、次に挙げる SOAP サービス定義である WSDL ファイルに従った SOAP 通信を行います。Tomcat+Axis 以外の環境で、SOAP 形式のサーバを実装する場合は、この WSDL の定義に沿った Web サービスを実装してください。

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
```



```
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://webservice.pslx.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://webservice.pslx.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://webservice.pslx.org/">
      <s:element name="Request">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="target" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="xml" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="RequestResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="RequestResult"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="RequestSoapIn">
    <wsdl:part name="parameters" element="tns:Request" />
  </wsdl:message>
  <wsdl:message name="RequestSoapOut">
    <wsdl:part name="parameters" element="tns:RequestResponse" />
  </wsdl:message>
  <wsdl:portType name="ServiceSoap">
    <wsdl:operation name="Request">
      <wsdl:input message="tns:RequestSoapIn" />
      <wsdl:output message="tns:RequestSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Request">
      <soap:operation soapAction="http://webservice.pslx.org/Request"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
```



```
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Request">
      <soap12:operation soapAction="http://webservice.pslx.org/Request"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
    <wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address
location="http://localhost:4383/Pps_SimpleServer_Soap/Service.asmx" />
    </wsdl:port>
    <wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12:address
location="http://localhost:4383/Pps_SimpleServer_Soap/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

◆ HTTP POST によるサーバの構築

PPS メッセージサービスを利用して、HTTP POST 形式でメッセージ通信を行う場合のサーバ構築方法について説明します。ここでは、Tomcat および Servlet を用いて単純な動的 Web ページを作成し、要求メッセージに対する応答メッセージを返すサーバの構築例を挙げます。なお HTTP POST によるサーバは、Tomcat+Servlet に限らず、IIS+ASP.NET や Apache+PHP など構成で構築することができます。クライアントは、PPS メッセージサービスを通じて、ここで構築したサーバに対して HTTP POST 形式でメッセージを送受信します。

次に、HTTP POST に対応した Web サーバのひな形プログラムを示します。このひな形プログラムは、org.pslx.Messaging.HttpService パッケージの PpsMessaging.java に含まれています。ここでは、クライアントから送られてきた依頼メッセージを受け取り、処理した上でその返信メッセージを返すプログラムを実装してください。


```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.setCharacterEncoding("utf-8");

    String xml;
    String senderId;
    String sendTo;

    if("application/x-www-form-urlencoded".equals(request.getContentType())) {
        xml = request.getParameter("xml");
        senderId = request.getParameter("senderid");
        sendTo = request.getParameter("sendto");

    } else if("application/xml".equals(request.getContentType())) {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(request.getInputStream(), "utf-8"));
        StringBuffer buffer = new StringBuffer();
        String line;
        while ((line = reader.readLine()) != null) {
            buffer.append(line);
        }
        xml = buffer.toString();
        senderId = request.getHeader("X-SenderId");
        sendTo = request.getHeader("X-SendTo");

    } else {
        PrintWriter writer = response.getWriter();
        writer.write("リクエストが不正です。");
        writer.close();
        return;
    }

    response.setContentType("text/html");
    response.setCharacterEncoding("utf-8");
    PrintWriter writer = response.getWriter();
    writer.write("サーバ(HTTP POST経由)からの応答メッセージです。¥r¥n");
    writer.write(senderId + "から" + sendTo + "へ次のメッセージを受け取りました¥r¥n");
    writer.write(xml);
    writer.close();
}
```

doPost メソッドは、クライアントから PPS メッセージサービスによって送られたすべてのリクエストを処理します。上記のプログラムでは、ローカル変数 xml に、クライアントからの要求が含まれるメッセージが代入されます。そのメッセージに従って処理を行い、Response.Write メソッドで応答メッセージを出力してください。ローカル変数 senderId は、

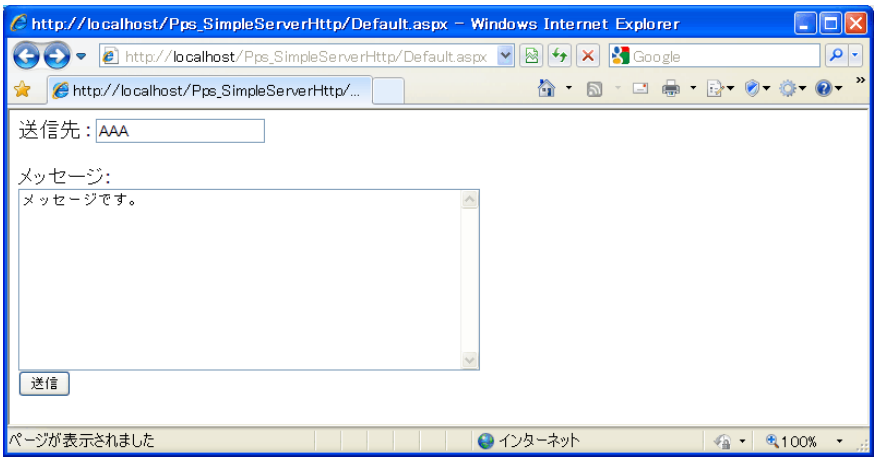
要求を送ったアプリケーションを識別するための名称が代入されます。またローカル変数 sendTo には、要求の送り先を表すコネクション名が代入されます。なお、PPS メッセージサービスの HTTP POST 形式によるメッセージ通信では、ContentType が“application/x-www-form-urlencoded”または“application/xml”のどちらかで送られます。サーバは、どちらの ContentType でも受け付けることができるように実装してください。

実際に業務アプリケーション開発において、要求メッセージを処理し応答メッセージを返すサーバを実装する場合には、doPost メソッドにおいて、PPS ドキュメントサービスなどを用いて、処理を行うロジックを実装します。クライアントから受け取った XML メッセージの内容を解釈し、PPS の規約にしたがって、返信メッセージを出力してください。返信メッセージが必要ない場合には、長さ 0 の文字列を出力してください。

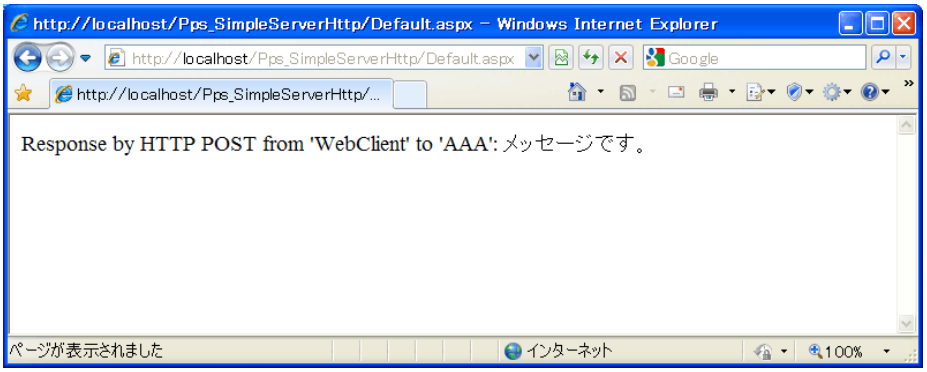
HTTP POST 形式に対応したサーバの場合は、PPS メッセージサービスからに限らず、Web ブラウザからメッセージを送ることができます。ここでは、作成した動的 Web ページを Web ブラウザからアクセスした場合に、サーバに対してメッセージを送ることができるように doGet メソッドで次のような HTML を出力するように書きます。

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType("text/html; charset=UTF-8");

    PrintWriter writer = response.getWriter();
    writer.write("<html>¥n");
    writer.write("<head>¥n");
    writer.write("<meta http-equiv=¥"Content-Type¥" content=¥"text/html;
charset=windows-31j¥">¥n");
    writer.write("<title>HTTP/POSTサンプル</title>¥n");
    writer.write("</head>¥n");
    writer.write("<body>¥n");

    writer.write("<form action=¥"PpsMessaging¥" method=¥"POST¥">¥n");
    writer.write("<input type=¥"hidden¥" name=¥"senderid¥" value=¥"WebClient¥"/>¥n");
    writer.write("送信先 :¥n");
    writer.write("<input type=¥"text¥" name=¥"sendto¥" value=¥"AAA¥"/><br>¥n");
    writer.write("<textarea name=¥"xml¥" cols=¥"50¥" rows=¥"10¥"></textarea>¥n");
    writer.write("<input type=¥"submit¥" value=¥"送信¥">¥n");
    writer.write("</form>¥n");
    writer.write("</body>¥n");
    writer.write("</html>¥n");
    writer.write("");
    writer.close();
}
```


上記の Servlet を Tomcat でデプロイし、アクセスすると、HTTP のサンプルにあるプログラムを実行すると、次のような送信用のフォームが表示されます。



ここで、任意のメッセージを入力し、送信ボタンを押すと、Web ブラウザがサーバに対して POST 形式でメッセージを送信され、サーバから次のような応答メッセージが得られます。



◆ XML スタイルシートによる表示

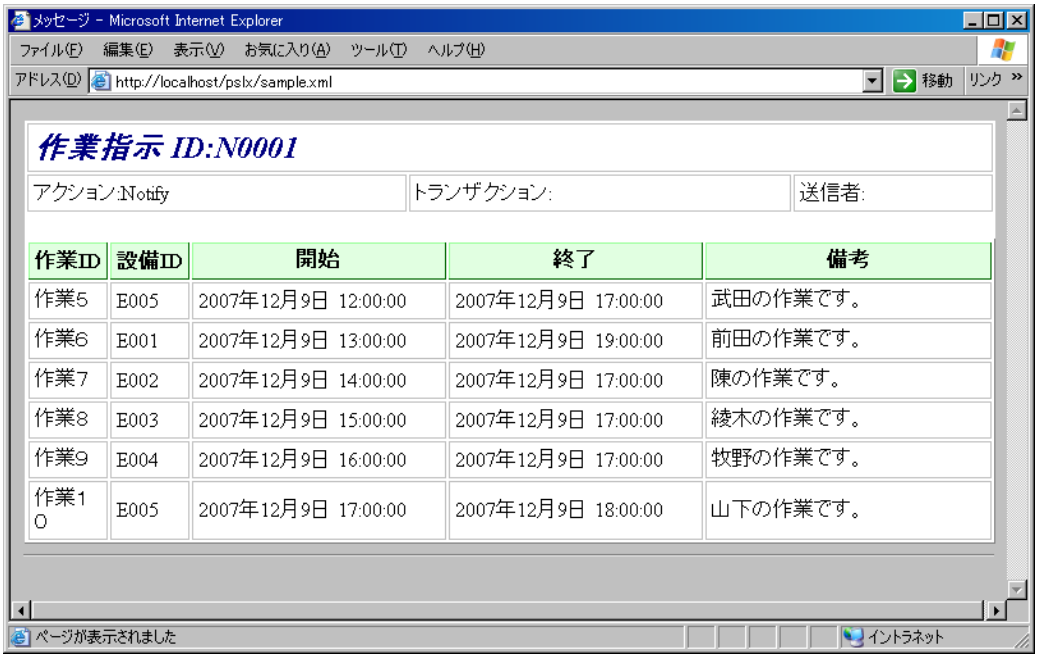
Web ブラウザをクライアントとして使用する場合は、提供されている共通コンポーネントをクライアントで使用することができません。Web ブラウザをクライアントとして使う場合は、一般に、XML スタイルシート(XSLT)を用いて XML 形式の業務ドキュメントを、帳票形式などに変換し、ブラウザに表示する方法が考えられます。ここでは、XSLT を用いて XML 形式の業務ドキュメントを表形式で表示するサンプルを示します。

この場合、返信メッセージに XML スタイルシートの指定が Web サーバによって挿入さ

れることになります。スタイルシートを任意に切り替えるために、必要に応じて POST メッセージに引数を追加することも可能です。

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pslx-style.xsl"?>
<Message id="0" sender="DefaultSender" xmlns="http://docs.oasis-open.org/pps/2009">
...以下省略
```

Web ブラウザには、次のような帳票が表示されます。Web ブラウザ上に表示された表は、上記の XML を XML スタイルシートによって整形し表示したものです。




```
<xsl:template match="pslx:Document[@name='OperationSchedule']">
  <div class="doc">
    <table>
      <tr>
        <td colspan="4">
          <span class="title">
            作業指示 ID:<xsl:value-of select="@id"/>
          </span>
        </td>
      </tr>
      <tr>
        <td>
          アクション:<xsl:value-of select="@action"/>
```



```
          </td>
          <td>
            トランザクション:<xsl:value-of select="@transaction"/>
          </td>
          <td>
            送信者:<xsl:value-of select="@sender"/>
          </td>
        </tr>
      </table>
      <p>
      </p>
      <table border="1">
        <tr>
          <th>作業ID</th>
          <th>設備ID</th>
          <th>開始</th>
          <th>終了</th>
          <th>備考</th>
        </tr>
        <xsl:for-each select="pslx:Operation">
          <tr>
            <td>
              <xsl:value-of select="@id" />
            </td>
            <td>
              <xsl:value-of select="pslx:Assign[@type='pps:equipment']/@resource" />
            </td>
            <td class="datetime">
              <xsl:apply-templates select="pslx:Start" />
            </td>
            <td class="datetime">
              <xsl:apply-templates select="pslx:End" />
            </td>
            <td class="remark">
              <xsl:value-of
                  select="pslx:Description[@type='pps:comment']/pslx:Char/@value" />
            </td>
          </tr>
        </xsl:for-each>
      </table>
    </div>
    <hr />
  </xsl:template>
```


付録　サンプル実装プログラム

PPS メッセージングサービス Java 版のコンポーネントには、サンプルプログラムが含まれています。本マニュアルで解説した内容に合わせて、次のサンプルプログラムが用意されています。これらのサンプルは、Java 環境で動作確認することができます。なお通信方式としてメッセージキューを利用する場合は、ActiveMQ や WebSphereMQ などへの接続に必要なパッケージが必要です。各メッセージキューの JMS を使用した接続方法に関するドキュメントをご覧ください。これらのサンプルプログラムが実際のアプリケーションプログラムの開発にあたり、参考になれば幸いです。

プログラム名	概要
SimpleSender.java	任意のテキストメッセージを送信するアプリケーションプログラムのサンプルです。
SimpleReceiver.java	任意のテキストメッセージを依頼したタイミングで受信するアプリケーションプログラムのサンプルです。
SimpleReceiverAsync.java	任意のテキストメッセージを受信したタイミングで受取るアプリケーションプログラムのサンプルです。
SimpleClient.java	任意のテキストメッセージを送信し、その直後に返信を受取るアプリケーションプログラムのサンプルです。
SimpleClientAsync.java	任意のテキストメッセージを送信した後、その返信を受信したタイミングで受取るアプリケーションプログラムのサンプルです。
SimpleServer.java	任意のテキストメッセージを受信し、送信者に対して返信メッセージを送信するアプリケーションプログラムのサンプルです。
PpsMessagingHttpServiceJava	Servlet(Tomcat)で動作する Web アプリケーションプログラムです。クライアントが送信した依頼メッセージに対して、返信メッセージを返します。HTTP の POST メソッドを利用しています。
PpsMessagingSoapServiceJava	SOAP サーバ(Tomcat Axis)で動作する Web アプリケーションプログラムです。クライアントが送信した依頼メッセージに対して、返信メッセージを返します。

