WS-Notificationの基礎

Webサービスの非同期メッセージングの実装

丸山不二夫 | (稚内北星学園大学)

前回説明したWS-RF(WS-ResrouceFramework)では，Webサービス間の非同期のメッセージングを，WS-Notification[1]という技術で実現しています．WS-Notificationはいくつかの仕様からなるのですが，基本的な部分は"WS-BaseNotification"という仕様にまとめられています．WS-Notificationには，この"WS-BaseNotification"のほかに，メッセージを分類するTopicの構造を規定した"WS-Topics"，メッセージの送り手と受け手の間に介在する「ブローカ」の役割を規定した"WS-BrokeredNotification"という2つの仕様があります．今回は，基本的な"WS-BaseNotification"の仕様を紹介したいと思います(**図-1**)．

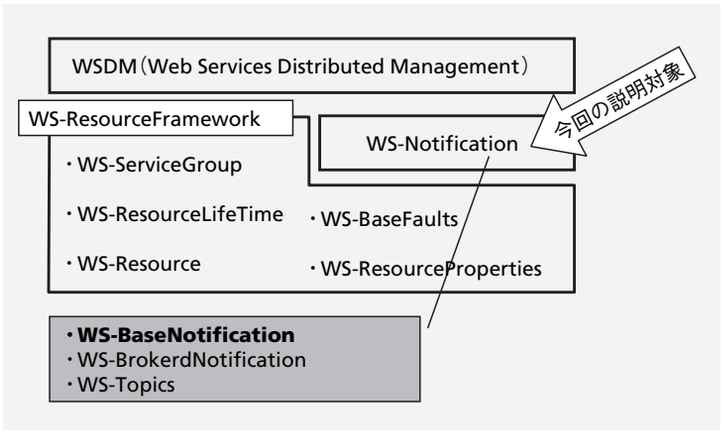

図-1 WS-RF/WSDMスタック

本稿では，まず，WS-Notificationの概観を与えようと思います．ここでは，受け手がSubscribe(登録)して，送り手がNotify(通知)するという関係が基本です(**図-2**)．注意してほしいことは，受け手側のポートにNotifyオペレーションが実装され，送り手側のポートにSubscribeオペレーションが実装されるということです．

続いて，こうした基本的な関係がWS-Notificationの仕様の中でどのように形式的に定義されているかを見ていきます．前回紹介したWSDLと，その中で利用されているWS-RFのリソースプロパティのスキーマ定義を見ることが，とても重要です．実は，WS-Notificationは，WS-RFとは独立の仕様なのですが，内部でWS-RFのリソースを操作する機能を利用しています．

最後に，サーバ上のリソース(WS-Resource)であるカウンタの値が変更されたときに，クライアントにそのことをコールバックで通知する，GT4(Globus Toolkit4)[2]のCounterサンプルを紹介したいと思います．このプログラムは，Javaで書かれていて，JavaでのWS-Notificationの実装のスタイルを学ぶことができます．

WS-Notificationを概観する

WS-Notificationはパブリッシュ／サブスクライブ(パブ／サブ)型のメッセージ通信に分類されます．パブ／サブ型の通信には基本的なパターンがあります．第1に，メッセージの送り手と受け手の2つの役割の区別があって，両者の間をやりとりされるメッセージがあります．これは，当然です．第2に，送り手の側にメッセージの受け手と受け手の興味の対象であるトピックの登録が必要です．そうしないと，送り手は，誰にどのメッセージを送ればいいのか分からなくなります．そして登録は，受け手の側の仕事です．第3に，送り手のメッセージ送信は，メッセージを受け取る受け手の側の，あるメソッド／オペレーションの呼び出しとして行われます．

具体的にWS-Notificationを見てみましょう．第1に，WS-Notificationでは，メッセージの送り手は「NotificationProducer」と，メッセージの受け取り手は「NotificationConsumer」と呼ばれています．これらは，いずれもWebサービスとして実装されています．メッセージは，「NotificationMessage」と呼ばれています(図-2)．

第2に，WS-Notificationでは，メッセージの送り手にメッセージの受け手を登録することを，「Subscribe」

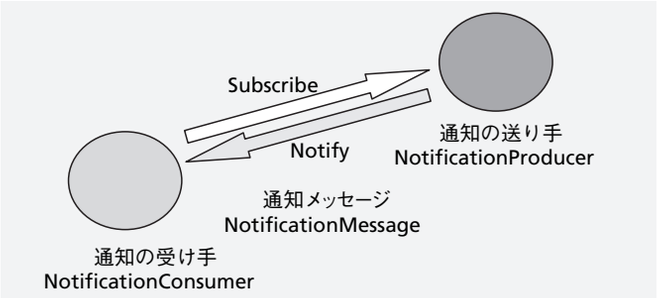

図 -2　WS-Notification の基本的プレーヤたち

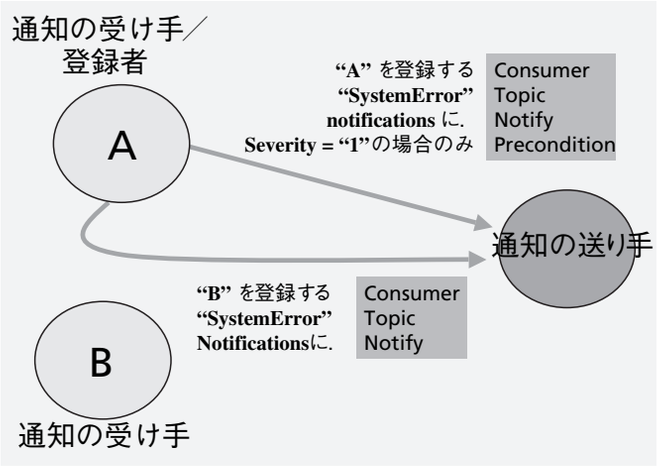

図 -3　WS-Notification Subscribe

するといいます．実際には，Web サービスとしての NotificationProducer は，そのポートに Subscribe オペレーションを実装することになります．WS-Notification では，Notification メッセージが，いくつかのテーマ（Subject）によってグループ分けされています．このグループは「Topic」と呼ばれています．「Subsctiption（登録）」の情報には，たとえば「“A”（Consumer）を登録する，“SystemError”（Topic）notifications に．Severity = “1” の場合のみ（Precondition）」という形で，Topic の情報が含まれています．Topic という用語・概念は，WS-Notification 以外でも，メッセージングの世界ではよく使われているものです（**図 -3**）．

第 3 に，WS-Notification では，メッセージの送り出しは，「Notify」すると呼ばれています．Notification Consumer は，そのポートに Notify オペレーションを実装していますが，Producer から Consumer へのメッセージの送り出しとは，Producer 側からの Consumer ポートの notify オペレーションの呼び出しにほかなりません（**図 -4**）．

NotificationProducer の WSDL 定義

NotificationProducer の WSDL での定義[3] を見てみることにしましょう（リスト 1）．Producer では，Subscribe オペレーションが定義され，NotificationProducerRP というリソース・プロパティが定義されていることが分かります．

リスト 1：NotificationProducer の WSDL 定義

```
<wsdl:portType name="NotificationProducer"
  wsrp:ResourceProperties =
              "wsnt:NotificationProducerRP">
   ........
   <wsdl:operation name="Subscribe">
      <wsdl:input  message=
              "wsnt:SubscribeRequest" />
      <wsdl:output message=
         "wsnt:SubscribeResponse" />
      <wsdl:fault name="ResourceUnknownFault"
                 message="wsnt:ResourceUnknownFault" />
      <wsdl:fault name="SubscribeCreationFailedFault"
                 message=
                  "wsnt:SubscribeCreationFailedFault"/>
      <wsdl:fault name="TopicPathDialectUnknownFault"
                 message=
                  "wsnt:TopicPathDialectUnknownFault"/>
   </wsdl:operation>
   ........
</wsdl:portType>
```

◉ Subscribe request のスキーマ定義

NotificationProducer ポートタイプの Subscribe オペレーションが，どのようなドキュメントをやりとりするのかを，先の WSDL 定義の types 節から見ておきましょう（リスト 2）．クライアントからサーバに対して向けられるリクエスト，すなわち NotificationConsumer から NotificationProducer に渡されるリクエスト・ドキュメントは，次のようなスキーマで定義されています．要素の細かな説明は省きますが，これらの要素のうち必須なのは，Consumer のエンドポイントリファレンスと TopicExpression だけであることに注目してください．リスト 3 に，Subscribe メッセージのサンプルを示します．

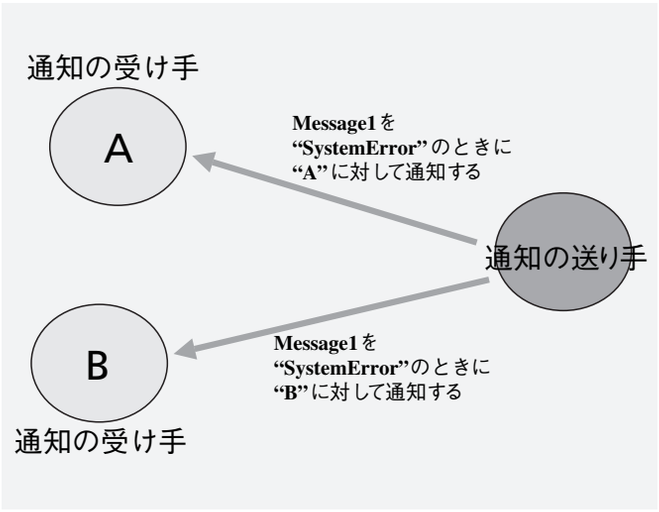

図 -4　WS-Notification Notify

リスト 2：Subscribe リクエストのスキーマ定義

```
<xsd:element name="Subscribe" >
    <xsd:complexType>
        <xsd:sequence>
            <!-- Consumer のエンドポイントリファレンス -->
            <xsd:element name="ConsumerReference"
                type="wsa:EndpointReferenceType"
                minOccurs="1" maxOccurs="1" />
            <!-- Producer がサポートしている Topic の指定 -->
            <xsd:element name="TopicExpression"
                type="wsnt:TopicExpressionType"
                minOccurs="1" maxOccurs="1" />
            <!-- Notify メッセージを使うか否か？ -->
            <xsd:element name="UseNotify"
                type="xsd:boolean" default="true"
                minOccurs="0" maxOccurs="1" />
            <!-- Notification の前提条件の設定 -->
            <xsd:element name="Precondition"
                type="wsrp:QueryExpressionType"
                minOccurs="0" maxOccurs="1" />
            <!-- NotificationMessage の選択 -->
            <xsd:element name="Selector"
                type="wsrp:QueryExpressionType"
                minOccurs="0" maxOccurs="1" />
            <!--  Subscription のポリシー -->
            <xsd:element name="SubscriptionPolicy"
                type="xsd:anyType"
                minOccurs="0" maxOccurs="1" />
            <!-- Subscription の生存期間の指定 -->
            <xsd:element name="InitialTerminationTime"
                type="xsd:dateTime"
                minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

リスト 3：Subscribe メッセージ

```
<wsnt:Subscribe>
  <wsnt:ConsumerReference>
    <wsa:Address>
      http://localhost/services/notifyReceiver
    </wsa:Address>
  </wsnt:ConsumerReference>
  <wsnt:TopicExpression dialect=
  "http://www.ibm.com/xmlns/stdwip/web-services/
WS-Topics/TopicExpression/simple">
       SystemError
  </wsnt:TopicExpression>
  <wsnt:UseNotify>true</wsnt:UseNotify>
</wsnt:Subscribe>
```

◉ Subscribe レスポンスのスキーマ定義

こうしたリクエストに対して，NotificationProducer は，リスト 4 のスキーマで定義されたレスポンスを返します．これは，Producer 内にリソースとして登録された Subscription 情報のエンドポイントリファレンスだと考えることができます．

リスト 4：Subscribe レスポンスのスキーマ定義

```
<xsd:element name="SubscribeResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="SubscriptionReference"
                type="wsa:EndpointReferenceType"
                minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

NotificationProducer の SubscriptionManager のポートタイプ

NotificationProducer には，リスト 5 の SubscriptionManager ポートタイプが定義されています．オペレーションとしては，wsrp:ResourceProperties ポートから GetResourceProperty オペレーションを，wsrl:ImmediateResourceTermination ポートから Destroy オペレーションを，wsrl:ScheduledResourceTermination から SetTerminationTime オペレーションを継承しています．また，このポートの独自のオペレーションとして，PauseSubscription と ResumeSubscription の 2 つが定義されています．

リスト 5：SubscriptionManager の WSDL 定義

```
<wsdl:portType name="SubscriptionManager"
     wsrp:ResourceProperties ="wsnt:SubscriptionManagerRP">

<!-- === extends wsrp:ResourceProperties ============= -->
    <wsdl:operation name="GetResourceProperty" .... />
<!-- === extends wsrl:ImmediateResourceTermination === -->
    <wsdl:operation name="Destroy" .... />
<!-- === extends wsrl:ScheduledResourceTermination === -->
    <wsdl:operation name="SetTerminationTime" .... />

<!-- === SubscriptionManager specific operations ===== -->
    <wsdl:operation name="PauseSubscription" .... />
    <wsdl:operation name="ResumeSubscription" .... />
</wsdl:portType>
```

◉ SubscriptionManager ポートタイプのリソースプロパティ

この SubscriptionManager ポートの特徴は，wsnt:SubscriptionManagerRP という名前の，リソースプロパティを持っていることです．そのスキーマ定義をリスト 6 に記します．注意してほしいことは，このリソースプロパティの要素が，先に見たリスト 1 の NotificationProducer の Subscribe オペレーションのリクエストメッセージに対応していることです．NotificationProducer への Subscription は，こうして，SubscriptionManager ポートのリソースプロパティ wsnt:SubscriptionManagerRP として，貯えられているのです．

リスト 6：Resource Properties for SubscriptionManager

```
<xsd:element name="SubscriptionManagerRP" >
    <xsd:complexType>
        <xsd:sequence>
            <!-- From WS-ResourceLifetime
            ScheduledResourceTermination -->
            <xsd:element ref="wsrl:CurrentTime"
                minOccurs="1" maxOccurs="1" />
            <xsd:element ref="wsrl:TerminationTime"
                minOccurs="1" maxOccurs="1" />

            <!-- SubscriptionManager specific  -->
            <xsd:element ref="wsnt:ConsumerReference"
                minOccurs="1" maxOccurs="1" />
            <xsd:element ref="wsnt:TopicExpression"
                minOccurs="1" maxOccurs="1" />
            <xsd:element ref="wsnt:UseNotify"
                minOccurs="1" maxOccurs="1" />
            <xsd:element ref="wsnt:Precondition"
                minOccurs="0" maxOccurs="1" />
            <xsd:element ref="wsnt:Selector"
                minOccurs="0" maxOccurs="1" />
            <xsd:element ref="wsnt:SubscriptionPolicy"
                minOccurs="0" maxOccurs="1" />
            <xsd:element ref="wsnt:CreationTime"
                minOccurs="0" maxOccurs="1" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

NotificationConsumerのWSDL定義

NotificationConsumer の WSDL での定義（一部）を見てみましょう（リスト 7）．Consumer では，Notify オペレーションが定義されています．同時に，このオペレーション定義では，wsdl:input 要素しかなく，wsdl:output 要素がないことに注意してください．これは，一方向の One-Way メッセージの operation 定義の特徴です．

リスト 7：NotificationConsumer

```
<wsdl:portType name="NotificationConsumer">
    <wsdl:operation name="Notify">
        <wsdl:input message="wsnt:Notify" />
    </wsdl:operation>
</wsdl:portType>
```

◉ NotificationConsumerの Notifyスキーマ定義

今度は，NotificationConsumer の Notify が，どのようなドキュメントを送り出すのか，そのスキーマ定義を見ておきましょう（リスト 8）．先にも触れたように，Notify オペレーションは，One-Way ですので，レスポンスはありません．

リスト 8：Notify スキーマ定義

```
<xsd:element name="Notify" >
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="NotificationMessage"
                type="wsnt:NotificationMessageHolderType"
                minOccurs="1" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

◉ NotificationMessageHolderType の スキーマ定義

もっとも，リスト 8 のスキーマ定義だけでは Notification メッセージの形は分かりません．リスト 9 に，NotificationMessageHolderType の定義を示します．これを見ると，Notification メッセージは，Topic とメッセージを発した NotificationProducer のエンドポイントリファレンスと，メッセージ本体の 3 つの部分から構成されていることが分かります．

リスト 9：NotificationMessageHolderType

```
<xsd:complexType name="NotificationMessageHolderType" >
    <xsd:sequence>
        <xsd:element name="Topic"
            type="wsnt:TopicExpressionType"
            minOccurs="1" maxOccurs="1" />
        <xsd:element name="ProducerReference"
            type="wsa:EndpointReferenceType"
            minOccurs="1" maxOccurs="1" />
        <xsd:element name="Message" type="xsd:anyType"
            minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>
```

リスト 10 に Notify メッセージの具体例を示します．

column

メッセージの向きはどう決まるか

一方向のメッセージングを WSDL で定義しようとするとき，難しいことが 1 つあります．それは，メッセージの向きをどう表現するのかということです．リスト 7 の NotificationConsumer ポートタイプでの Notify オペレーションを考えてみてください．どうして，ここのサブ要素は，output ではなく，input なのでしょう？　説明できますか？

1 つの問題は，Notify を「通知する」と考えると，それは送り手側のアクションであって，送り手側の operation を呼び出すことと考えやすいことです．そうではありません．Notify は，確かに送り手側から始まるアクションなのですが，それはネットワーク上で受け手側の Notify オペレーションを呼び出すことで実現されています．

ただ，そのことが分かったとしても，メッセージの向きをどう表現するかが決まるわけではありません．それは，次のようなルールで，約束事として決められています．

「ネットワーク上で，requestor と provider が相対しメッセージを交換するとき，メッセージの向きは，provider から見た向きで決める」

リスト 7 の Notify の例では，送り手が requestor，受け手が provider です．ですから，受け手側から見て，外から内に入ってくるメッセージだから，input だということになります．

コンピュータの世界では，理詰めで考えると理解できることが多いのですが，メッセージの向きは，前提となる約束事を知らないと，考えても理解できないという例の 1 つです．

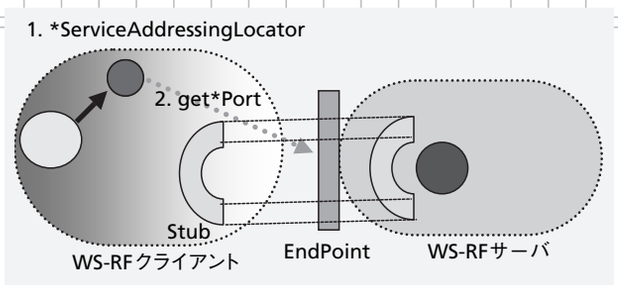

図 -5　ServiceAddressingLocator を生成し Endpoint に対して get*Port を実行すると Stub が獲得される

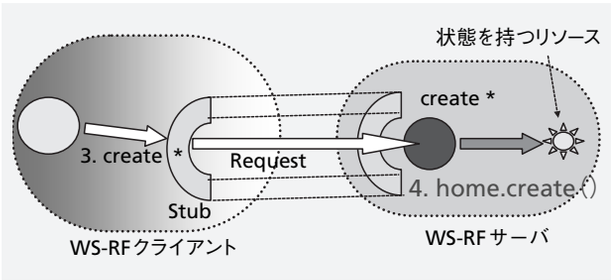

図 -6　Stub 上の create* は，メッセージを通じてサーバ上で create* を呼び出す

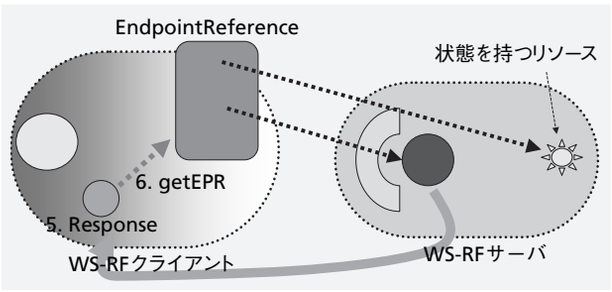

図 -7　Response に対する getEPR で，WS- リソースへの EPR が返る

リスト 10：Notify メッセージ

```
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:Topic>widget:TestTopic</wsnt:Topic>
    <wsnt:ProducerReference>
      <wsa:Address> http://localhost/services/
WidgetService</wsa:Address>
    </wsnt:ProducerReference>
    <wsnt:Message>test message</wsnt:Message>
  </wsnt:NotificationMessage>
</wsnt:Notify>
```

Counter サンプルを見る

ここでは，GT4 と一緒に提供されているサンプル・プログラムのうち，Counter というサンプルを取り上げたいと思います．Counter サンプルは，大きく 2 つの内容を持っています．1 つは，WS- リソースを生成しそれを管理する WS-RF です．Counter サンプルのもう 1 つの内容は，Notification の手法です．

◉クライアントからの WS- リソースの生成

前回は，WS-RF でのサーバ側のプログラミングについて見てきました．ここでは，WS-RF のクライアント側のプログラミングのスタイルを見ておこうと思います．WS-RF のプログラムでは，クライアント側からサーバ側に WS- リソースを生成するのは，基本的な手法です．

リスト 11 は，Counter というリソースを持つ，WS- リソースを生成するコードの一部です．**図 -5**，**6**，**7** を見てください．一般的には，次のような手順のプログラミングになります．

1. ServiceAddressingLocator のインスタンスを生成する（図 -5）
2. Endpoint に対して get*Port を行って，Stub を獲得する（図 -5）
3. Stub 上の create* は，サーバ上で create* を呼び出す（図 -6）
4. サーバ上の create* は Home クラスの create メソッドを通じてリソースを生成する（図 -6）
5. Stub 上の create* メソッドの呼び出しは，レスポンスを返す（図 -7）
6. Response から EndpointReference を獲得する（図 -7）

リスト 11：クライアントからの WS- リソースの生成

```
// 1. ServiceAddressingLocator のインスタンスを生成する
CounterServiceAddressingLocator locator =
          new CounterServiceAddressingLocator();

// 2. Endpoint に対して get*Port を行って，Stub を獲得する
service =
    "http://localhost:8080//services/CounterService";
URL endpoint = new URL(service);
CounterPortType port =
    locator.getCounterPortTypePort(endpoint);

// 3. Stub 上の create* は，サーバ上で create* を呼び出す
// 4. サーバ上の create* は Home クラスの create メソッドを通じて
      リソースを生成する
// 5. Stub 上の create* メソッドの呼び出しは，レスポンスを返す
CreateCounterResponse createResponse =
        port.createCounter(new CreateCounter());

// 6. Response から EndpointReference を獲得する
EndpointReferenceType epr =
        createResponse.getEndpointReference();
```

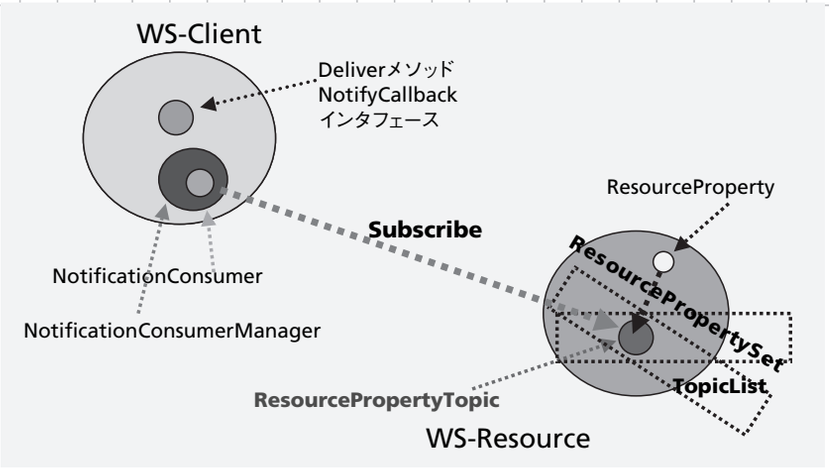

図 -8　TopicList と ResourcePropertySet への Topic の登録／ Subscribe

◉ リソースの変更がクライアントへの通知を引き起こす設定

Counter サンプルでは，サーバ側のリソースの値が変更されたときに，クライアント側に通知が発行されるようになっています．いくつかの部分に分けて，基本的な設定を見ていくことにしましょう．

◉ リソース TopicList への Topic の登録

サーバ側では，サーバが発行する Notification が属する Topic の管理を，管理すべき Topic インスタンスを TopicList に登録することで行っています．また，WS-RF の Java 実装では，サーバ側の WS- リソースは，ResourceProperty のインスタンスとして存在して，それらは，ResourcePropertySet に登録されて管理されています．要するに，あるサーバ（Producer）が，どのような Topic を扱っているかは，TopicList を見れば分かるし，また，そのサーバ上にどのような WS- リソース（ResourceProperty）が存在するかは，ResourcePropertySet を見れば分かるということです．

サーバリソース・プロパティの変更に反応するという ResourcePropertyTopic のインスタンスは，ResourceProperty のインスタンスから生成されます（リスト 12）．リスト 12 で少し分かりにくいのは，この ResourcePropertyTopic は，実は，ResourceProperty インタフェースを実装しており，Topic でもあると同時に ResourceProperty でもあるという，二重の性格を持っているということです．

ですから，リスト 12 で，ResourcePropertyTopic のインスタンスとして生成された value は，まず，ResourceProperty として ResourcePropertySet に登録され，つづいて Topic として TopicList に登録されています．ResourcePropertyTopic の二重の性格に対応して，同じインスタンスが異なる管理用のリストに，二重に登録されているわけです．ここでは，最後に，ResourceProperty としての値が初期化されています（リスト 12，**図 -8**）．

リスト 12：TopicList と ResourcePropertySet への Topic の登録

```
-------------------------------------------------------------------------
private ResourcePropertySet propSet;
private TopicList topicList;
......
......
this.value = new ResourcePropertyTopic(
    new SimpleResourceProperty(VALUE));
this.propSet.add(this.value);
this.topicList.addTopic((Topic) this.value);
this.value.add(new Integer(0));
-------------------------------------------------------------------------
```

◉ NotifyCallback インタフェースの実装と NotificationConsumerManager の利用

今度はクライアント側です．コールバックで通知を受け取るクライアントは，NotifyCallback インタフェースを実装する必要があります．具体的には，deliver メソッドを実装する必要があります．また，このクライアントを NotificationConsumer とするために，NotificationConsumerManager というクラスを利用した，リスト 13 のようなコードが使われています（図 -8）．

リスト 13：NotificationConsumerManager の利用

```
-------------------------------------------------------------------------
consumer = NotificationConsumerManager.getInstance();
consumer.startListening();
EndpointReferenceType consumerEPR =
    consumer.createNotificationConsumer
            (new CounterClient());
-------------------------------------------------------------------------
```

◉ Topic の設定と Subscribe

Topic を設定したら，subscribe を呼び出します（リスト 14）．これでパブ／サブ型通信の基本的な設定は終了です（図 -8）．

リスト 14：Subscribe

```
Subscribe request = new Subscribe();
request.setUseNotify(Boolean.TRUE);
request.setConsumerReference(consumerEPR);
TopicExpressionType topicExpression =
                          new TopicExpressionType();
topicExpression.setDialect
                 (WSNConstants.SIMPLE_TOPIC_DIALECT);
topicExpression.setValue(Counter.VALUE);
request.setTopicExpression(topicExpression);

EndpointReferenceType subscriptionEPR =
    counterPort.subscribe(request).
                          getSubscriptionReference();
```

◉ リソースの値の変更と Notification

カウンタに 3 を加算するコードは次のようなものです（リスト 15）．サービス・ロケータから，エンドポイント・レファレンスを指定してポート（Stub）を取得し，add メソッドを呼び出しています．その次のコードは，前回見た WS-RF の getResourceProperty メソッドを使ってリソースの取得を行っています．こうしてカウンタの値を確認することができます．

リスト 15：リソースの値の変更と Notification

```
CounterPortType addPort =
    locator.getCounterPortTypePort(counterEPR);
client.setOptions((Stub)addPort);
addPort.add(3);
......
......
GetResourcePropertyResponse getRPResponse =
    counterPort.getResourceProperty(Counter.VALUE);
System.out.println("Counter has value: " +
                   getRPResponse.get_any()[0].
                   getValue());
```

カウンタの値が更新されると，ResourceProperty Change が検知されて，クライアントに対して Notify が行われ，クライアントの deliver メソッドが呼び出されることになります（**図 -9**）．

だいぶ駆け足でしたが，WS-Notification の基本部分を見てきました．WS-Notification では，冒頭でも触れたように，今回登場した NotificationProducer と NotificationConsumer に加えて，両者の仲介をする Broker の拡張や，subscribe 時に利用される Topic が階層構造を持つなどの拡張がなされています．紙面の都合で省略しましたが今回の内容の詳細な説明を公開していますので参考にしてください [4]．

次回は，Web サービスの管理フレームワークを提供する WSDM について紹介したいと思います．

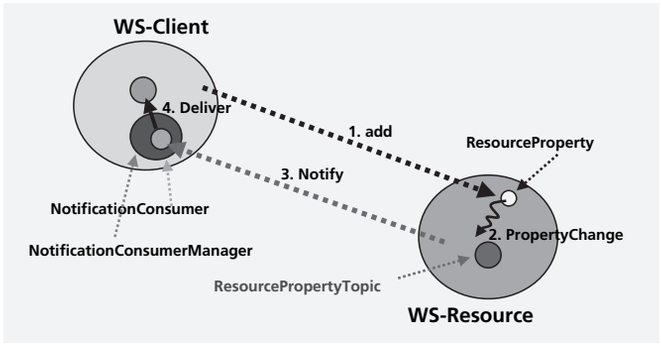

図 -9　PropertyChange / Notify

参考文献

1）OASIS WSN TC Web ページ：http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
2）Globus Toolkit Web ページ：http://www.globus.org/toolkit/
3）WS-BaseNotification v1.0：http://www-128.ibm.com/developerworks/webservices/library/specification/ws-notification/
4）サマースクール「Grid/WSRF とビジネスプロセスの統合」テキスト：http://www.wakhok.ac.jp/~maruyama/summer04/wsrf.pdf

（平成 18 年 11 月 7 日受付）

丸山不二夫（正会員）
maruyama@wakhok.ac.jp

東大教育学部卒業．一橋大学大学院社会学研究科博士課程修了．「最北端・最先端」をモットーに，稚内で新しいスタイルとコンテンツの情報教育を展開．「新しい時代の新しい大学」を目指して，社会人 IT 技術者をターゲットとしたサテライト校を秋葉原に設置．アジアでの IT 教育も熱心に展開している．現在，稚内北星学園大学学長．