WSDLとWS-ResourceFramework

Webサービスのインタフェース記述と実装の概要

丸山不二夫 (稚内北星学園大学)

WSDLとは何か

今回は，前回に引き続いて，グリッド由来の Web サービス標準技術である WS-ResourceFramework（WS-RF）の説明を行います．本論に入る前に，まず，WSDL（Web Services Description Language）について解説をします．今回のテーマである WS-RF の手法を具体的に理解する上で，WSDL の知識が不可欠だからです．WSDL は，現在の Web サービス標準技術の中で，最も重要な要素技術の 1 つです．今回の WS-RF だけではなく，この連載のほとんどすべての回の内容は，WSDL と関係があります（**図 -1**）．

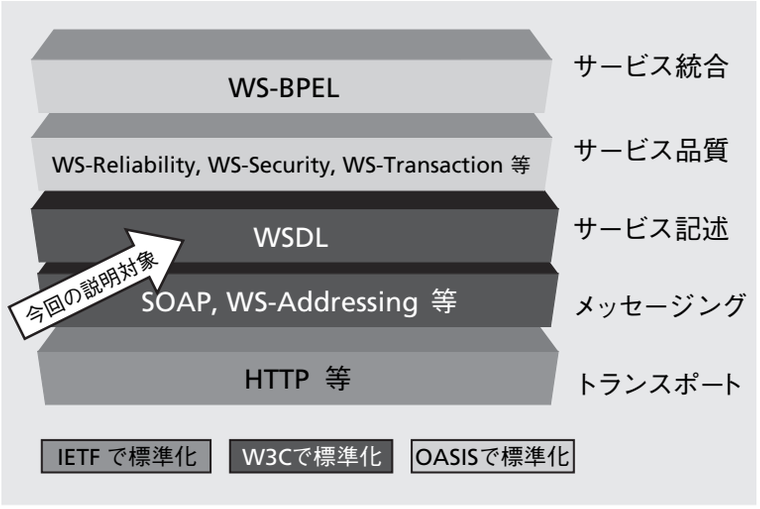

図 -1　今回説明する Web サービス標準

WSDL は，文字通り Web サービスを記述する言語です．「Web サービスを記述する」と書きましたが，WSDL は Web サービスそのものの「実装」を与える言語ではありません．Web サービスを定義するのに WSDL は必須ですが，WSDL が与えられれば，Web サービスのプログラムが動くわけではありません．WSDL は，Web サービスの「インタフェース」定義を与えます．Web サービスのプログラムを動かすためには，WSDL による「インタフェース」定義に加えて，具体的なプログラム言語，たとえば Java による，Web サービスの「実装」が必要になります．

Web サービスをインタフェースと実装に分けることによって，Web サービスのプログラミングは，ずいぶん簡単になります．WSDL2Java というプログラムは，与えられた WSDL から，この WSDL が定義する Web サービスの実行に必要な多数の補助的な Java クラスを生成します．プログラマは，Web サービスの中心的なロジックの部分だけを実装すればいいのです．逆に，Java2WSDL というプログラムは，Java のインタフェースから，対応する WSDL を生成してくれます．グリッドの代表的な実装である，GT4（Globus Toolkit 4）でも，これらの WSDL と Java の相互変換プログラムは活躍しています．

WSDL は，このようにプログラムによって読み込まれたり，プログラムによって生成されることが多いので，人間が読んだり書いたりするものではないと思われることがあります．ただ，筆者はそうは思っていません．少なくとも，WSDL がどのようにして Web サービスを表現しているかを理解することは，Web サービスの理解にとって，とても大事なことだと思っています．

まずは，HelloWorldのWSDLから

ここでは，maruyama という文字列を与えれば，"Hello maruyama" という文字列を返す，HelloWorld の Web サービス版の WSDL を見てみましょう．もっとも，こうした HelloWorld サービスの特徴は，実装によるものですので，ここでの WSDL には，文字列を与えれば文字列が返るという定義しか含まれていないことに注意してください．

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloWorld"
 targetNamespace="http://hello.jaxrpc.samples/"
 xmlns:tns="http://hello.jaxrpc.samples/"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap=
   "http://schemas.xmlsoap.org/wsdl/soap/">

  <types />

  <message name="sayHello">
    <part name="String_1" type="xsd:string" />
  </message>
  <message name="sayHelloResponse">
    <part name="result" type="xsd:string" />
  </message>

  <portType name="Hello">
    <operation name="sayHello"
        parameterOrder="String_1">
      <input message="tns:sayHello" />
      <output message="tns:sayHelloResponse" />
    </operation>
  </portType>

  <binding name="HelloBinding" type="tns:Hello">
    <operation name="sayHello">
      <input>
        <soap:body encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded"
        namespace="http://hello.jaxrpc.samples/" />
      </input>
      <output>
        <soap:body encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"
        use="encoded"
        namespace="http://hello.jaxrpc.samples/" />
      </output>
      <soap:operation soapAction="" />
    </operation>
    <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style="rpc" />
  </binding>

  <service name="HelloWorld">
    <port name="HelloPort"
         binding="tns:HelloBinding">
      <soap:address
      location="http://localhost:8080/axis/Hello" />
    </port>
  </service>
</definitions>
```

WSDLを構成する要素たち

WSDL を構成する要素たちを，最後に定義されている service 要素から，説明しましょう．後ろに出てくる要素が，前の要素を名前で参照していることに注意してください．

◉ service 要素

service 要素は，URL の location 属性を抱えた address 要素を含む，port 要素から構成されている，比較的，簡単な形をしています．この URL は，Web サービスを提供するサーバとサービスの Endpoint の指定です．このサービスの URL 以外の具体的な情報は，port 要素の binding 属性で参照されている，binding 要素の情報に記述されています．

◉ binding 要素

binding 要素は，結構，複雑な形をしています．もう一度確認してほしいことは，この binding 要素の name 属性は，先に見たように service 要素の port 要素の binding 属性で参照されており，また，この binding 要素の type 属性は，先行する portType の name 属性の値にほかなりません．次の SOAP の namespace を持つ binding 要素は，この Web サービスが RPC（Remote Procedure Call）スタイルのサービス呼び出しを行い，transport 層は，HTTP を使うことを意味しています．

```
<wsdlsoap:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
```

多くの Web サービスが RPC 型で，HTTP を使っているのですが，その場合には，この例に示すような形で，WSDL の内部で宣言されていなければなりません．ちなみに，WS-RF では，style="rpc" はなく style="document" を使っています．また，WS-RF は，use="encoded" ではなく，use="literal" を使っています．

style="rpc" の指定は，クライアントがメソッド名と引数をサーバに渡し，サーバがクライアントに，その計算結果の返り値を返すという，RPC スタイルで，メッセージを交換することを意味しますが，style="document" の指定は，クライアントとサーバがもっと一般的なスタイルで，双方が XML ドキュメントそのものをメッセージとして交換することを意味します．現在では，こうしたスタイルは，「ドキュメント・セントリック（中心）」のスタイルとして，その汎用性と解釈の自由さが注目さ

れています．

use="encoded" の指定は，SOAP に固有のエンコーディングを使うという指定なのですが，Web サービスの相互運用性を高めるために提案され，広く受け入れられている WS-I の規格では，SOAP エンコーディングを使わないことが推奨されています．use="literal" の指定は，SOAP 固有のエンコーディングではなく，より一般的な XML Schema で定義される XML の型システムを，そのまま利用しようということです．こうした Schema 定義は，WSDL の types 節に置かれることになります．

binding 要素の大部分は，operation 要素の定義にあてられています．この例では，sayHello という operation が定義されています．operation は，input と output の定義から構成されています．WSDL の operation は，Java のメソッドのプロトタイプ宣言に対応するものです．Input はメソッドの引数，output はメソッドの返り値と考えればいいと思います．この点では，次に見る，portType での operation の定義と共通しています．ただ，binding での operation の定義は，portType での operation の定義より，実装に近い具体的なものだと考えることができます．

◉ portType 要素

portType 要素は，operation を定義しています．binding 要素での operation の定義と portType 要素での operation の定義は，1 対 1 に対応しています．同一の operation を，binding と portType とは違ったレベルで定義しているのです．

portType では，operation の input と output は，SOAP メッセージの encodingStyle や namespace といった具体的な情報によってではなく，それを構成するメッセージそのもの，抽象的なレベルでのメッセージの型によって定義されています．operation の input と output は，RPC のメソッドの引数と返り値に対応しますので，portType 要素は，メソッドの引数と返り値に，それぞれの型を対応付けていると考えることができます．ですから，binding での operation の定義が，RPC 呼び出しのプロトコルや引数のエンコーディング方式のスタイルを規定しているのに対して，portType での operation の定義は，関数のプロトタイプ定義，メソッドのインタフェース定義に相当すると考えることが可能です．

◉ message 要素

message の名前は，portType での operation 定義の input，output の中で利用されます．message 要素は，part 要素を抱えています．そして part 要素は，名前属性と型属性を持っています．このサンプルでは，この型属性が，type="xsd:String" のような形で，schema の NameSpace を示す xsd で修飾されていることに注意してください．

◉ types 要素

先の例では定義が空でしたが，WSDL の中で大事な働きをする要素が 1 つ残っています．それが，types 要素です．types 要素は，XML のドキュメントの型の定義を与える XML Schema を利用して新しい型を定義して，その型を WSDL で使うことを可能とします．types 要素と message 要素は，新しい型のワイアリング・フォーマット，すなわち，ネットワーク上に流れるメッセージのフォーマットを規定します．

WS-リソースをWSDLで表現する

WSDL の解説を終えて，本題の WS-RF に入りたいと思います．まず前回の復習ですが，WS- リソースというのは，「状態を持たない Web サービス」と「状態を持つリソース」を分離したうえで，組み合わせたものです．今回は，この両者が，どのように結び付けられているかを見てみましょう．

WS-RF の Java 実装では，リソースとは Java のあるインスタンスにほかなりません．問題は，このリソース・インスタンスの状態は，外部とは，「リソース・プロパティーズ・ドキュメント」という，XML ドキュメントを通じてのみ，アクセスされるということです（**図 -2**）．XML ドキュメントは，リソース・インスタンスそのものではありませんが，その「表現」と考えればいいと思います．この例では，GenericDiskDriveProperties という「リソース･プロパティーズ･ドキュメント」が，リソース･インスタンスの状態の「表現」として描かれています．リソース・インスタンスの状態の変化はその表現の変化を引き起こし，表現の変更はリソース・インスタンスの変化を引き起こします．両者は，対応付けられなければならないのですが，Java のインスタンスと XML のドキュメントは別のものですから，意識的な同期が必要になります．

WS-RFでのWSDLの拡張

それでは，このリソースの表現である「リソース・プロパティーズ・ドキュメント」は，Web サービス

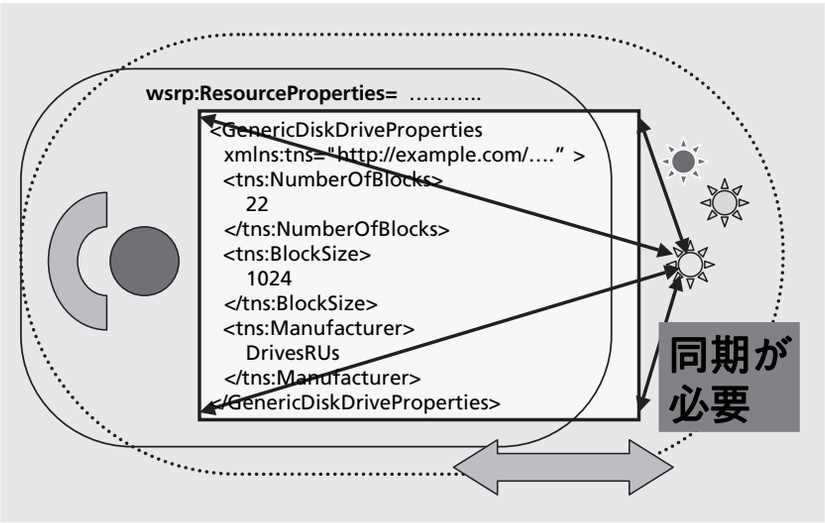

図 -2　状態を持つリソースの「表現」としての WS- リソース・プロパティーズ・ドキュメント

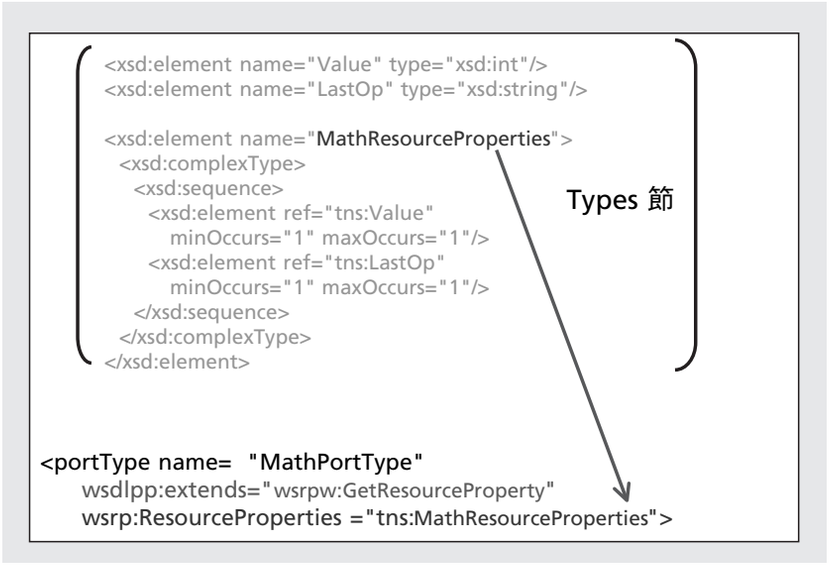

図 -3　WSDL portType でのプロパティ定義

とどのように関連付けられているのでしょうか？　実は，WS-RF では，そのドキュメントの型が，WSDL の portType の ResourceProperties 属性で定義されることになっています．同時に，そのリソース・プロパティーズの，スキーマ定義は，WSDL の type 節におかれなければなりません（**図 -3**）．このように，リソースを扱う WS-RF では，それに対応した WSDL の拡張が行われることになります．

ここでは，portType 要素の 2 つの属性，wsdlpp:extends と wsrp:ResourceProperties に注目してください．

後者の ResourceProperties 属性は，WS-RF に固有の拡張で，先に見たように，WS- リソースの表現であるリソース・プロパティーズ・ドキュメントを指定するのに使われます．

前者の extends 属性は，もう少し一般的な WSDL の拡張です．基本的には，Java の Interface 定義での extends と同じです．extends 以下で指定される portType を含んだ WSDL を見つけ出して，その portType に関連した WSDL の内容をすべて含んだ WSDL をつくるのだと思ってください．こうして，extends を使えば，いちい

PortType	Operation	Provider
GetResourceProperty	GetResourceProperty	GetRPProvider
GetMultipleResourceProperties	GetMultipleResourceProperties	GetMRPProvider
SetResourceProperties	SetResourceProperties	SetRPProvider
QueryResourceProperties	QueryResourceProperties	QueryRPProvider

表 -1　WS-ResourceProperties.wsdl の portType と operation

PortType + ResourceProperty	Operation	Provider
ScheduledResourceTermination + ScheduledResourceTerminationRP	SetTerminationTime	SetTerminationTimeProvider
ImmediateResourceTermination	Destroy	DestroyProvider

表 -2　WS-ResourceLifetime.wsdl の portType と operation

ち同じ WSDL の記述を繰り返さないでも，それとの差分を記述することで，簡単に WSDL を書くことができます．こうした，portType 要素へ extends を導入することは，portType 自身の名前を interface に改めた新しい WSDL2.0 では，インタフェースの継承・拡大として，明確に位置付けられることになりました．

WS-RF が提供するサービス

WS-RF の本体を構成するサービス群は，まず，WSDL の形で提供されています．また，WS-RF のシステムは，WSDL だけではなく，WSDL が定義する WS-RF が提供するサービスを，具体的に実装していなければなりません．Operation ごとに用意された，これらのサービスの提供者を，GT4 では，Provider と呼んでいます．こうした Provider のサービスは，もちろん，対応する WSDL から生成されたものです．

ここでは，WS-RF の基本的な WSDL ファイルを紹介して，どのようなサービスが，WS-RF ではあらかじめ用意されているのかを見てみることにしましょう．

◉ WS-ResourceProperties.wsdl

この WSDL は，WS- リソースへアクセスするサービスを提供します．WS-RF にとっては，最も基本的なサービスになります（**表 -1**）．それぞれのリソースは，リソース・プロパティーズ・ドキュメント中の，リソースプロパティとしてアクセスされることになります．その限りでは，リソースとリソースプロパティは，同じ意味で利用されています．

GetResourceProperty は，リソースの名前を 1 つ指定して，その値を獲得します．GetMultipleResourceProperties は，複数のリソースの名前を指定して，それらの複数の値を獲得します．SetResourceProperties は，リソースの変更なのですが，Insert，Update，Delete という 3 つの操作のいずれかを選ぶことができます．

QueryResourceProperties は，XPATH 等の Query 言語を通じて，リソース・プロパティーズ・ドキュメントに直接問合せを行います．

詳しいことは次回に説明しますが，WS-BaseN.wsdl が定義する WS-Notification というサービス群を利用して，それぞれのリソース・プロパティごとにリソースの変化を通知するサービスを作ることができます．この時には，リソースの新旧の値のペアからなる，ResourcePropertyValueChangeNotification メッセージが発行されることになります．

以下に見る，すべての WS-RF のサービス群で，portType にリソース・プロパティの定義を持つもの，すなわち，サービス固有のリソースを持つものは，そのリソースの変化に対して，このメカニズムを利用して，通知を行うことが可能です．

◉ WS-ResourceLifetime.wsdl

この WSDL は，リソースの生存時間の管理を行うサービスを定義しています（**表 -2**）．SetTerminationTime は，リソースの終了時間を指定します．Destory は，その場で即座にリソースを破棄する時に使います．

リソースの終了時に，通知を発行することが可能です．

◉ WS-ServiceGroup.wsdl

この WSDL の提供するサービスは，複数の異なった

PortType + ResourceProperty	Operation	Provider
ServiceGroup + ServiceGroupRP	GetResourceProperty	GetRPProvider
ServiceGroupEntry + ServiceGroupEntryRP	GetResourceProperty	GetRPProvider
ServiceGroupRegistration + ServiceGroupRP	GetResourceProperty Add	GetRPProvider AddProvider

表 -3　WS-ServiceGroup.wsdl の portType と operation

Web サービスたちを，1 つのグループにまとめることを可能にします（**表 -3**）．まとめられた ServiceGroup は，それ自身 1 つの Web サービスとして，エンドポイント・リファレンス（EPR）を持ちます．

ServiceGroupEntryRP を構成するのは，この ServiceGroup の EPR と，そのグループのメンバになった Web サービスの EPR のペアです．ServiceGroupRP は，こうした Entry のリストからなる WS- リソースだと考えればいいと思います．

ServiceGroupRegistration ポートタイプの Add オペレーションは，ServiceGroup に，メンバを追加・登録する時に利用されます．

ServiceGroup が変更・抹消，あるいは，個別の Entry の追加・消去といったリソースの変化について，通知を発行することができます．

◉ WS-BaseN.wsdl

この WSDL は，通知のサービスを提供します．厳密には，このサービス群は，WS-RF には含まれない，独自のサービス群を形成しています．しかし，上でも見たように，それぞれのリソースの変更について通知を発行するという想定は自然なものですので，WS-RF と非常に近い関係を持ったサービス群と考えていいと思います．

このサービス群については，次回に，詳しく説明したいと思います．

WS-RFのサーバ側のプログラミング（1）
– まずは，WSDLの作成から –

これから，WS-RF での具体的なサービス構築の手順の紹介を通じて，WS-RF のプログラミング・スタイルを見ていきたいと思います．本稿では，グリッドの標準的な開発ツールである，GT4（Globus Toolkit 4）での開発スタイルに準拠して解説を進めたいと思います．

WS-RF のサーバ側のプログラミングでは，WS- リソースの設計をするのが最初になります．WS-RF では，WS- リソースに対する基本的な操作等が，あらかじめ，いくつかの Web サービスとして与えられています．これらのサービス群が，WS-RF の中核になります．WS-RF でプログラムするということは，これらの WS-RF の提供するサービスを利用して，新しいサービスを作ることにほかなりません．

もちろん，WS-RF が提供するサービスの定義は，先に見たように WSDL で与えられています．また，これらのサービスを利用して作られたサービスも，WSDL で記述されることになります．その意味では，WS-RF のサーバ側のプログラミングには，WS-RF が提供する WSDL を利用して，別の WSDL を作成する手順が含まれることになります．

1. まず，リソースを表現する XML ドキュメントのスキーマ定義を作成します．これらのスキーマのインスタンスとしての XML ドキュメントが，それぞれのリソースの「表現」になります．
2. ついで，そのスキーマ定義を types 節に持ち，かつ，その名前を portType の ResourceProperties 属性に持つ，WSDL を作成します．
3. もし，サーバ側のサービスが，提供されている WS-RF のサービスを利用するのなら，その操作を含む portType を，作成するサービスの WSDL の portType の extends 属性の指定を通じて取り込みます．

こうした WSDL 作成の手順を，2 つの例で見てみましょう．

プログラムは，いずれも，WS- リソースとしての整数 Value の値を変更する簡単なものです．最初の例は，具体的な数値と "plus"，"minus" といった演算命令を与えて，その命令に基づいて Value の値を変更するものです．このとき，直前の演算命令もリソースとして記憶されます．2 つ目の例は，与えられた整数値での Value のカウントアップしかできないカウンタの例です．ただし，カウントアップのたびに，状態が変わったという通知を

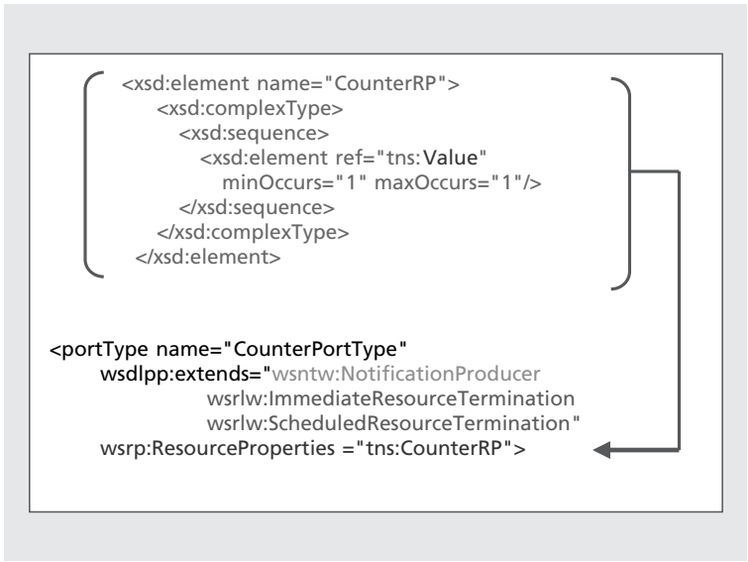

図 -4　WSDL portType でのプロパティ定義

発行します．両者の，WSDL の定義を見てみることにしましょう．

図 -3 では，内部に int 型の Value と String 型の LastOp を抱えた，MathResourceProperties というスキーマが types 節に置かれて，それが ResourceProperties 属性に指定されています．また，ここでは，GetResourceProperty という portType を extends していますが，この portType は，WS-RF が利用する基本的な WSDL の 1 つである，WS-ResourceProperties.wsdl という WSDL ファイルの中に含まれています．

図 -4 では，内部に int 型の Value を抱えた，CounterRP というスキーマが ResourceProperties 属性に指定されています．また，NotificationProducer と ImmediateResourceTermination と ScheduledResourceTermination という 3 つの portType が extends されています．NotificationProducer という portType は，WS-BaseN.wsdl という WSDL に含まれており，ImmediateResourceTermination と ScheduledResourceTermination という 2 つの portType は，ResourceLifetime.wsdl という WSDL に含まれています．

WS-RF でのプログラミングは，その提供するサービスの WSDL 定義から始まるところが，プログラミング・スタイルの大きな特徴になっています．

WS-RFのサーバ側のプログラミング（2）– Javaでのサービスの実装 –

ただし，WSDL でのサービスの定義は，あくまでも，サービスのインタフェースの定義にほかなりません．実際に動くサービスを作るためには，WSDL の作成だけでは不十分で，サービスの実装が必要となります．WS-RF の実装は，Java をはじめいくつかの言語で可能です．紙幅の都合で，サービスの実装については，細かな説明には立ち入りません．

ここでは，Java でサービスを実装する場合，どのようなクラスが必要になるかを簡単に整理したいと思います．

- まず，リソースの定義がサービス側で必要になります．リソース定義クラスには，リソースプロパティセットを生成し，リソースプロパティを操作するメソッドが含まれます．
- さらに，サービスを定義するクラスが必要です．サービス定義クラスには，リソースを生成するメソッド，リソースの状態を操作するメソッドが含まれます．
- また，WS-RF では，Home クラスが，リソース生成時にサービスから呼び出されて一意な ID を生成し，サービスとリソースをつなぐ役割を果たします．

サービスの実装は，Resource と ResourceProperties という 2 つの Java のインタフェースの実装として行わ

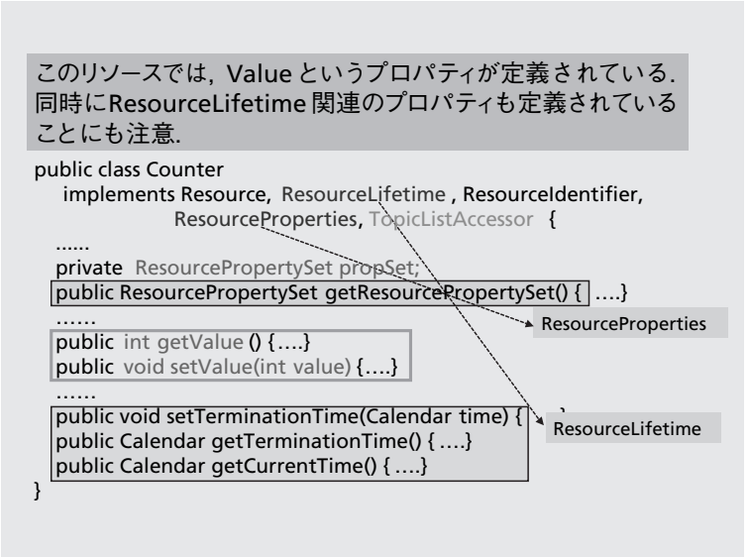

図 -5　リソースクラスでのプロパティ定義

れます．リソースプロパティーズを公開するリソースは，ResourceProperties インタフェースを実装することが求められます．いったん，コンストラクタ内の操作がすめば，リソースの値へのアクセスは，簡単な setter/getter を通じて可能になります．

図 -5 は，図 -3 の WSDL に対応する，Counter リソースの定義クラスの一部です．このリソース・クラスのプロパティ定義を中心に見ています．この例では，Counter リソースは，ResourceProperties インタフェースのほかに，Resource, ResourceLifetime, ResourceIdentifier, TopicListAccessor といったインタフェースを持っていることに注意してください．これらのインタフェースに対応するプロパティが導入され，それらにアクセスする setter/getter の実装が行われています．

WS-RFのサーバ側のプログラミング（3）
–デプロイ・ファイルの設定と，WSDLからの補助クラスの生成–

ここまで，WS-RF でのサービスのインタフェースを定義する WSDL と，サービスを実装する Java のクラスたちの作り方を見てきました．ただ，それだけだと不十分で，こうして定義されたサービスを，サーバ上にデプロイするための情報が必要になります．GT4 では，Web サービスのデプロイに一般的に利用されている，deploy-service.wsdd ファイルと deploy-jidi-config.xml ファイルをそのまま利用します．Counter サービスのデプロイ・ファイルの一部です．

```
-----------------------------------------------------------------------

<deployment name="defaultServerConfig" .... >

    <service name="CounterService"
      provider="Handler"
        use="literal" style="document">
        .....
        <parameter name="className"
            value="org.globus.wsrf.samples.counter.
                    CounterService"/>
         <wsdlFile>share/schema/core/samples/
          counter/counter_service.wsdl</wsdlFile>
        <parameter name="scope"
         value="Application"/>
        <parameter name="providers" value="
            DestroyProvider
            SetTerminationTimeProvider
            GetRPProvider
            QueryRPProvider GetMRPProvider
            SubscribeProvider GetCurrentMessage
            Provider"/>
    </service>
    .....
    .....
</deployment>

-----------------------------------------------------------------------
```

ここでは，name="providers" を持つ parameter 要素の value 属性に注目してください．これらのプロバイダは，図 -3 で extends された portType に属する

Operation のプロバイダに対応していることを確認してください．新しいサービスは，portType の extends を通じて，これらのサービスを利用するように作られているわけですので，新しいサービスの実行環境へのデプロイの際に，これらのサービス・プロバイダをデプロイするのは当然のことです．

これでようやく準備が整いました．GT4 では，これらのファイルを全部そろえておけば，あとは，GT4 のシステムが，ant コマンド 1 つで，すべての必要なファイルを生成し，かつそれらを，1 つの GAR ファイルに束ねる作業を行ってくれます．そこで中心的な役割を果たしているのは，extends を用いて簡潔に表現された WSDL を，extends を含まない長い WSDL に書き換える作業（flattering と呼ばれています）と，そうして生成され WSDL から，WSDL2Java を使ってサービス提供に必要な Java クラスを生成する作業です．

次回予告

今回は, 拡張された WSDL を利用した, WS-RF でのサーバ側のプログラミングについて，主に見てきました．次回は，WS-RF を利用したメッセージングのメカニズムを規定した WS-Notification の仕様を紹介したいと思います．あわせて，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いこうと思います．

（平成 18 年 10 月 6 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．

column

WS-ResourceFramework から WS-ResourceTransfer へ

今年の 3 月 15 日，HP, IBM, Intel, Microsoft の 4 社は，共同で "Toward Converging Web Service Standards for Resources, Events, and Management" という文書を発表します．そこでは，現在，Web サービス上で，リソースやイベントや管理を扱う WS-* 仕様が，複数存在しているが，それを近い将来には 1 つのものに整理・統合したいということが述べられていました．Grid の世界の反応ですが，Globus の Ian Foster は，こうした動きを基本的には歓迎するというコメントを発表しています．それが事実なら，Grid の世界で標準的に利用されている WS-ResourceFramework は WS-ResourceTransfer へ，WS-Notification は WS-EventingNotification へと変わることになります．WS-RF と WS-N をベースにしている，WSDM（Web Services Distributed Management）も新しいものに置き換わることになると思います．

ここでは，WS-ResourceTransfer のベースになっている，WS-Transfer の特徴を見ておきましょう．WS-Transfer は，2 つの実体を対象にしています．1 つは「リソース」で，それはエンドポイント・リファレンスでアドレス指定されて，XML の「表現」を持ちます．もう 1 つの実体は, リソースを生成する Web サービスである「リソース･ファクトリ」です．

WS-Transfer では，GET, PUT, DELETE というオペレーションが定義されていて，リソースに対する，獲得・更新・削除が可能となり，リソース・ファクトリに対する CREATE オペレーションで，リソースの生成が可能になります．

WS-Transfer には，リソースの表現として WSDL で定義されるべきリソースプロパティはなくなります．なぜなら，WS-Transfer では，アドレス可能なリソースがすべて表現を持つとされているからです．それに伴って，リソースプロパティに対する GetResourceProperty，SetResourceProperty という操作は，リソースに対する直接の，GET, PUT, DELETE といった操作に変わります．筆者は，こうした動きには，前号のコラムで見た REST の考え方の強い影響を感じています．