WS-AddressingとWS-ResourceFramework

状態を持つリソースを，状態を持たないサービスでどう扱うか？

丸山不二夫 | （稚内北星学園大学）

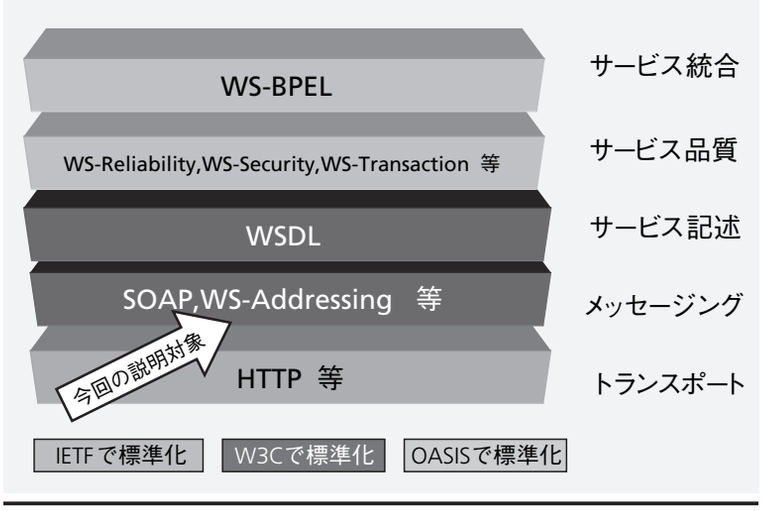

図 -1 今回説明する Web サービス標準

WS-AddressingとWS-ResourceFrameworkの関係

今回と次回は，Web サービスの新しい展開の中で，その役割がますます重要なものになりつつある，Web Service Addressing（WS-Addressing）[1] と WS-ResourceFramework[2]（以下，WS-RF と略）標準技術について紹介したいと思います．紙面の都合上 WS-RF は，次回第 3 回目で詳しく説明します．

連載の第 1 回目で紹介しましたが，WS-Addressing は Web/XML 関連の基本標準を策定する W3C の標準であり，WS-RF は Web/XML に関する応用的な標準を策定する OASIS の標準です．今回の解説の中心となる WS-Addressing 標準の位置付けを**図 -1** に示します．WS-Addressing の仕様策定には IBM，Microsoft，BEA，SAP 各社等が，WS-RF の仕様策定には IBM，富士通各社等がかかわっています．現在でも仕様の改良，他の仕様との統合などが検討されているのですが，最新動向については次回少し解説したいと思います．

さて WS-Addressing（以下，WS- アドレッシング）と WS-RF の関係ですが，名前から分かるとおり，前者は Web サービスのリクエスタ（クライアント）とプロバイダ（サーバ）間でやりとりする SOAP（Simple Object Access Protocol）メッセージを適切に送受信等するためのアドレス仕様です．後者は Web サービスプロバイダ上のコンテナ等に生成されるリソースを管理するための仕様です．Web サービスプロバイダにメッセージが届くとサーバ上では何らかの処理が実行されます．何回かメッセージをやりとりして 1 つのまとまった処理が完結する Web サービスを提供しようとした場合，サーバ側ではリソースの状態を管理し受け取ったメッセージと状態を保持するリソースとを関連づける仕組みが必要になってきます．このような仕組みを実現するのに必要な標準仕様が WS- アドレッシングと WS-RF なのです．少し詳しく見ていきましょう．

WS-アドレッシングとは何か？

ではまず WS- アドレッシング標準を紹介したいと思います．WS- アドレッシングは，基本的に「エンドポイント・リファレンス」と「メッセージ情報ヘッダ」という 2 つの技術要素から構成されています．

前者の「エンドポイント・リファレンス」は，これまで Web サービスの呼び出しに利用されていた Web サービスのアドレスを示す単純な URL を，「URL ＋アルファ」のかたちで拡張したものです．WS- アドレッシング技術の典型的な利用例として，前回見た Globus のツールキット GT4 の中で，グリッドの基礎技術として利用されている WS-RF があります．WS-RF では，「URL ＋アルファ」のアルファの部分に，「状態を持つリソース」についての情報を埋め込みます．WS-RF についての詳しい説明は次回に行いますが，WS-RF のうち WS- リソースと呼ばれる「状態を持つリソース」の考え方については，この回で少し詳しく説明したいと思います．

後者の「メッセージ情報ヘッダ」も，やはり Web サービスの拡張を目指したものです．これまでの Web サー

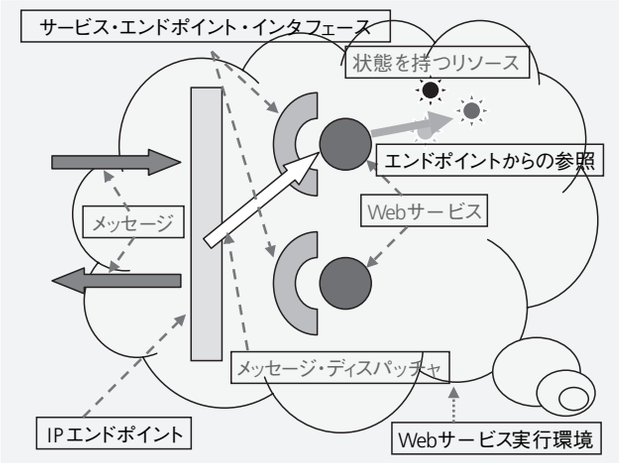

図 -2　Web サービスと WS- アドレッシング

ビスでは，HTTP プロトコル以外の方法で，メッセージをやり取りすることを想定していませんでしたが，WS-アドレッシングでの「メッセージ情報ヘッダ」の導入は，Web サービスのメッセージ交換を任意のトランスポート層へ拡大することを可能とするものです．

図で見る WS- アドレッシング

図 -2 を見てください．この図は，Web サービスのサーバ側の働きを図示したものです．

全体が，「Web サービス実行環境」というコンテナの内部に入っています．Web サービスはこの「実行環境」に配備された 1 つのコンポーネントです．

コンテナの前面に立っているのは，このコンテナの「エンドポイント」です．IP ネットワーク的には，IP アドレスとポート番号のペアからなるこのエンドポイントが，このコンテナのネットワーク・アドレスを代表しています．Web サービスの一般的な URL は，http://IPAddrress:port/AnyPath という形式をしているのですが，URL の先頭部分を構成するのはこの IP エンドポイントの情報です．

Web サービスでは，URL のこの情報の後ろの AnyPath パス情報は，図でのメッセージ・ディスパッチャで解釈され，ディスパッチャは目的の Web サービスにメッセージを渡すのですが，このパスつきの URL で指定される Web サービスのインタフェースを，「サービス・エンドポイント・インタフェース」と呼びます．Web サービスでは，これを省略して「エンドポイント」と呼ぶことがあります．Web サービスで，普通，エンドポイントというと，一般的には，IP ネットワーク的なエンドポイントではなく，こちらの方のエンドポイントを指すことに注意してください．

この図には，Web サービスと「状態を持つリソース」との関係も示されています．Web サービスの 1 つから「状態を持つリソース」への矢印が 1 つのびていますが，こうした参照を可能にするのが，「エンドポイント・リファレンス」です．サービスは，メッセージの処理の際，関連する「状態を持つリソース」と関係を持つかもしれません．「エンドポイント・リファレンス」の導入は，クライアントが Web サービスの呼び出しの際，こうしたサーバ側の要素にアクセスすることを可能にします．

通常の Web サービスでは，メッセージのやり取りは HTTP プロトコルで行われます．「メッセージ情報ヘッダ」の導入は，このエンドポイントが，トランスポート層を HTTP 以外のプロトコルで送られてきたメッセージを受け取ることを可能にします．この時，内部の Web サービスが理解可能なフォーマットにメッセージは変形され，サービスに送りつけられます．もちろん，処理された結果は，レスポンスとして返されます．

「エンドポイント・リファレンス」の導入の背景

WS- アドレッシングでの「エンドポイント・リファレンス」の導入の背景を見ていきたいと思います．そのためには，WS-RF という技術と WS- アドレッシングという技術との関係を見るのがいいと思います．両者は，別々の技術ですが，WS- アドレッシングの最初の応用例の 1 つが WS-RF であったという関係で結びついています．また，両者は「状態を持つリソースを，状態を持たないサービスでどう扱うか？」という共通の問題意識の中から生まれたと考えることができます．WS-RF の考え方の基礎にあるのは，状態を持つリソースをどのように Web サービスでモデル化するのかという問題なのです．

ただ，直接この問題に対する WS-RF の解答を見る前に，少し回り道をしたいと思います．といいますのは，ネットワーク上のリソースとその状態，サービスとその状態という問題群は，昔から存在していたもので，WS-アドレッシングや WS-RF が初めて取り上げたものではないからです．ここでは，そうしたもう少し一般的な視点から，この問題を考えたいと思います．

WS-RF でのこうした問題への対応は，少し後の「WS-リソースとは何か？」の章以降で説明します．

状態を持つリソースとサービス

「状態を持つリソース」ということでは，コンピュー

タのメモリやファイル・システムやデータベースに蓄えられている情報をイメージしてもらえばいいと思います．これらの情報は，コンピュータの働きによって，時間とともに変化します．こうした情報を「リソースの状態」と呼んでいるだけです．その意味では，すべての実用的なアプリケーションは，「状態を持つリソース」に関係していると考えても間違いではありません．

リソースが状態を持つのと同様に，サービスも状態を持つことがあります．あるサービスが，先行する処理の結果に応じて次の処理を変えるなら，そのサービスは状態を持つと考えることができます．たとえば，あるインターネット・モールに入って，そこで買い物をしたとしましょう．ショッピング・カートに欲しい品物を追加していって，あるいは，買うのをやめて品物を元に戻して，最後に料金を精算するわけですが，こうしたサービスは，「カートに入れられた商品」という状態を持ったサービスだと考えていいでしょう．ここでも，ほとんどすべてのサービスは，状態を持っているように見えます．

状態を持たないサービス

ただ，問題は少し複雑です．たとえば，先の例ですが，HTTP サーバの基本機能（たとえば，FORM や CGI）だけでショッピング・カートを作ろうとしてみれば，そのことはよく分かります．たとえ同じユーザが，HTTP サーバに連続してアクセスしたとしても，HTTP サーバは，そのユーザが前回のアクセスの際にカートに入れたものを覚えておいてはくれません．そうした情報をサーバ側に覚えさせるためには，工夫が必要になります．初心者はてこずるでしょう．

それには理由があります．HTTP サーバは，特定のリクエストに対して特定のレスポンスを返すのが基本動作です．サーバは，レスポンスを返すまでの間は，リクエストの情報を保持し続けるのですが，そうした情報はリクエスト・レスポンスのサイクルが一巡すると，サーバからは捨てられて，サーバは新しいリクエストを受け取る状態に戻ります．こうして，HTTP サーバは，常に同一の状態でリクエストを待ち続けます．その意味では，HTTP サーバは，基本的には，状態を持たないといっていいのです．

同一のユーザが，ある時間のあいだ同じサーバに継続してアクセスする時，それを「セッション」と呼ぶことがあります．アトミックな動作を繰り返すばかりで，直前の処理の情報さえも記憶しない HTTP サーバに，どのようにして，連続したセッションの同一性を意識させるのか？　あるいは，状態を持たない HTTP サーバに，セッション中のショッピング・カートのような「状態」を，どのようにして記憶させるのか？　Web アプリケーションにとっては，こうした問題は，実践的には重要な問題でした．

Cookie を使ったり，Servlet にセッションを管理させたり，あるいは，Stateful Session Beans を利用したりと，Web アプリケーションの技術的な進歩の 1 つの側面は，それ自体は状態を持たない HTTP サーバに，どのようにして状態を持つセッションを可能にするのかというものだと考えることもできます．

前回，見たように，Web アプリケーションと Web サービスは異なったものです．ただし，HTTP サーバだけでは状態を持たないという点は，両者に共通して当てはまります．すなわち，リクエストに，呼び出すべきメソッドの名前と引数を与え，レスポンスとしてサーバ上で計算された返り値を送り返すという，RPC（Remote Procedure Call）型で HTTP を利用する Web サービスは，やはり，状態を持たないサービスということになります．

状態を持たないサーバの利点

サーバにとって，それが状態を持たないことは，決して悪いことではありません．状態を持たないサーバのほうが状態を持つサーバよりシンプルな構造を持ちます．また，サーバがいったんトラブルで倒れても，状態を持たないサーバなら，再度立ち上げなおせば，それで処理を問題なく再開できます．状態を持たない Web サーバは，単純で，それゆえ，頑丈なサーバなのです．

それに対して，状態を持つサーバでは，ダウンした時点の状態を復元しないと正しい処理が行えなくなります．もちろん，必要に応じて状態を持つサーバも現実に利用されています．データベースサーバは，ある意味で，典型的な状態を持つサーバです．データベースサーバは，サーバの状態が不整合をおこさないように，コミットやロールバックといったトランザクションの管理機能を備えています．

グリッドの世界でも，Globus が WS-RF 以前に採用していた OGSI（Open Grid Services Infrastructure）という技術は，状態を持つサービスの一種でした [4]．

WS- リソースとは何か？

WS-RF という技術の中核にあるのは，「WS- リソース」という概念です．WS- リソースは，文字通り前半の WS

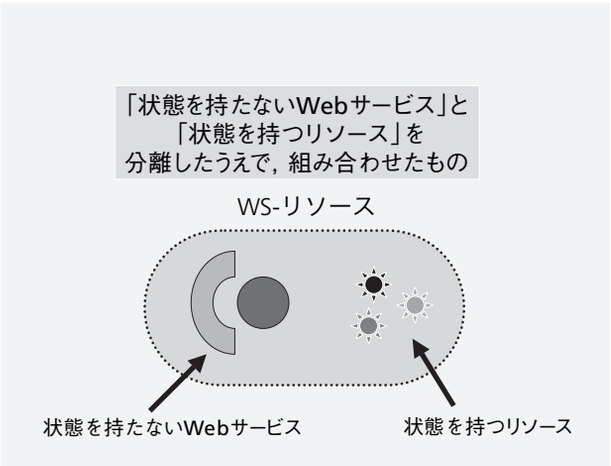

図 -3　WS- リソースとは？

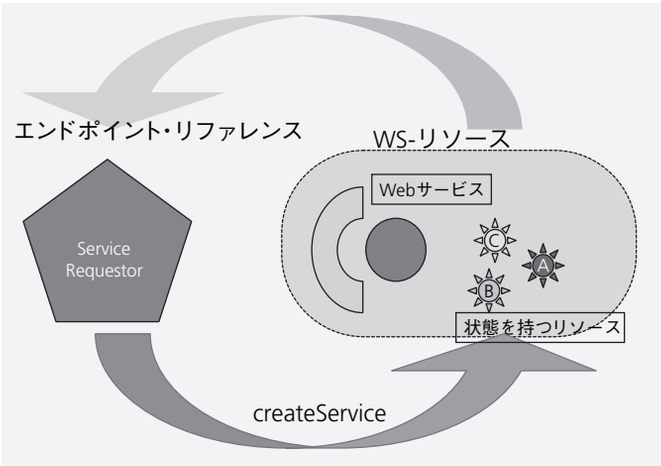

図 -4　WS- リソースの生成でエンドポイント・リファレンスが返る

すなわち Web サービスと，後半のリソースとの組合せです．ポイントは，前者は状態を持たず，後者は状態を持つということです．WS- リソースとは，「状態を持たない Web サービス」と「状態を持つリソース」とを，いったん分離したうえで，組み合わせたものです．WS-RF とは，2 つの構成要素を持つ構造物である WS- リソースを扱うフレームワークにほかなりません（**図 -3**）．

WS-アドレッシングとWS-リソース

図 -2 を再度見てください．ここでは，ネットワーク上で同一のエンドポイント・アドレスのもとに，異なる複数の Web サービスや複数の状態を持つリソースが存在することに注意してください．従来の Web サービスは，エンドポイントのアドレスのみでサービスを一意に指定できたのに対して，図 -2 ではコンテナの内部に複数の Web サービスや複数のリソースを含んでいるので，リソースを含めたアドレスの指定が必要となります．「WS- アドレッシング」は，あるエンドポイントに配備された Web サービスや WS- リソースのアドレスを表現する能力を持つように，Web サービスのアドレス仕様を拡張したものです．

WS- アドレッシングでのアドレス仕様の拡張は，「エンドポイント・リファレンス」という概念の導入によって行われます．少し単純化していうと，エンドポイント・リファレンスは，Web サービスのアドレスとリソースへの参照ポインタという 2 つのアドレスを持っています．WS- アドレッシングは，WS- リソースとそれを操作するフレームワークである WS-RF にとって不可欠なだけではなく，Web サービスを組み合わせて業務フローを記述する OASIS 標準，WS-BPEL（Web Service Business Process Execution Language）などにも利用され，Web サービスの拡張にとって必須の基礎技術になっています．具体的には，「エンドポイント・リファレンス」は，XML で表現されます．

WS-RF では，サーバに対して新しいサービスとリソースを生成せよという，たとえば createService リクエストを発行して，WS- リソースをサーバ側に生成します．この時，WS- リソース生成の結果として「エンドポイント・リファレンス」がレスポンスとして返されます（**図 -4**）．

WS-RF では，WS- リソースがすでにサーバ側に生成されている場合には，条件を指定してそうしたリソースを検索することができます．検索の結果，ある WS- リソースが見つかった場合には，この WS- リソースをポイントしている「エンドポイント・リファレンス」が返されます．このように，あるメソッドが，Web サービスや WS- リソースへの参照を返す場合には，この「エンドポイント・リファレンス」が利用されます．

「エンドポイント・リファレンス」を構成するもの

実際のエンドポイント・リファレンスの例を見てみましょう．**図 -5** を見てください．この例は，先の createService リクエストの結果，WS- リソースが生成され，返って来たエンドポイント・リファレンスを示したものです．エンドポイント・リファレンスには，Address コンポーネントと ReferenceProperties コンポーネントという 2 つの重要なコンポーネントが含まれていることが分かると思います．

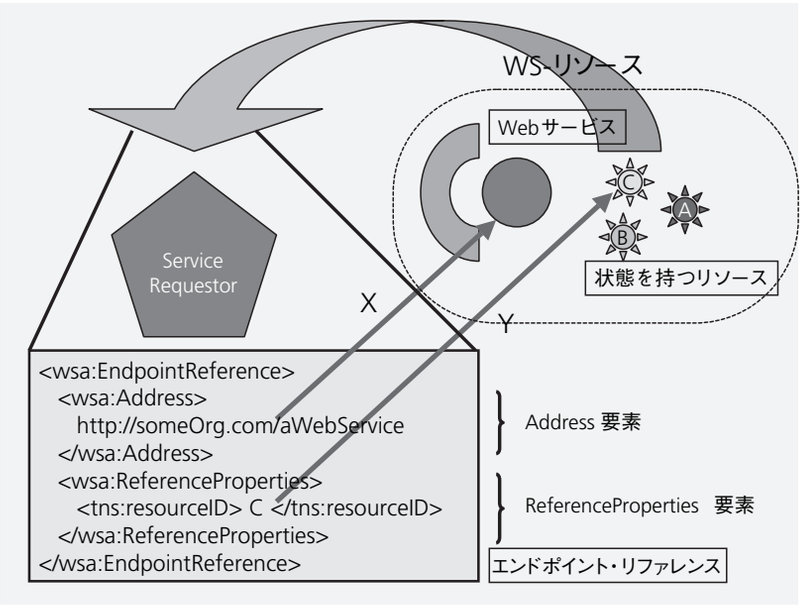

図 -5　図 -4 で返されるエンドポイント・リファレンスの詳細

Address コンポーネントは，Web サービスのネットワークのトランスポート層に特有なサービスのエンドポイント・アドレスをポイントしています（図 -5 ポインタ X）．HTTP ベースのトランスポートの場合には，それは，エンドポイントの URL です．前回も触れましたように Web サービスのインタフェースを記述する言語として WSDL（Web Service Description Language）がありますが，WSDL において Web サービスのアクセスポイントを指定する役割を持つ service 節の port 要素の内部に表れるのは，このアドレスです．

ReferenceProperties コンポーネントは，冒頭の部分で「URL ＋アルファ」と述べたアルファの部分の情報を抱えています．この例では，Address コンポーネントがポイントするエンドポイントに配備された WS- リソース ID の XML での表現を含んでいます．この WS- リソース ID は，エンドポイントのもとにある Web サービスが，リクエスト・メッセージを実行する際に，コンテキストとして利用する WS- リソースをポイントしています（図 -5 ポインタ Y）．

WS-アドレッシングの構成要素：「エンドポイント・リファレンス」

これまでの Web サービスでは，エンドポイントの情報は，先にも述べたように WSDL の service 節の port 要素（address 要素の location 属性）に記述されていました．ただ，こうしたエンドポイントの記述はスタティックなものです．現実には，Web サービスのインスタンスは，状態を持つさまざまな作用の結果として，ダイナミックに生成され，状態が変化することがあります．このときエンドポイントの情報もその記述もダイナミックに生成され変更されなければなりません．

エンドポイント・リファレンスは，論理的には WSDL の仕様を拡張したものですが，それを置き換えるものではありません．ただ，状態を持つサービスのある特定のインスタンスに名前をつけて識別する場合とか，そのインスタンスに特有な状態を他に伝え，あるいは，そうした状態を動的に変更する場合などに，エンドポイント・リファレンスは，WSDL の service 節の代わりに利用されます．

もう少し分かりやすいかたちで表すと次のようになります．

```
---------------------------------------------------------------------
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceProperties> ...
  </wsa:ReferenceProperties> ?
  <wsa:PortType>xs:QName</wsa:PortType> ?
  <wsa:ServiceName PortName=
  "xs:NCName"?>xs:QName</wsa:ServiceName> ?
  <wsp:Policy/> *
</wsa:EndpointReference>
---------------------------------------------------------------------
```

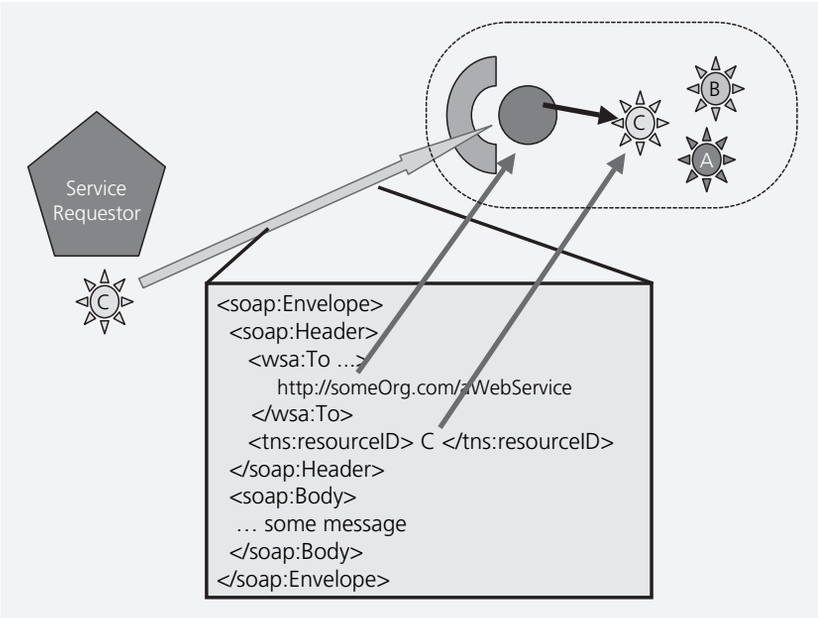

図 -6　クライアントから WS- リソースを参照する

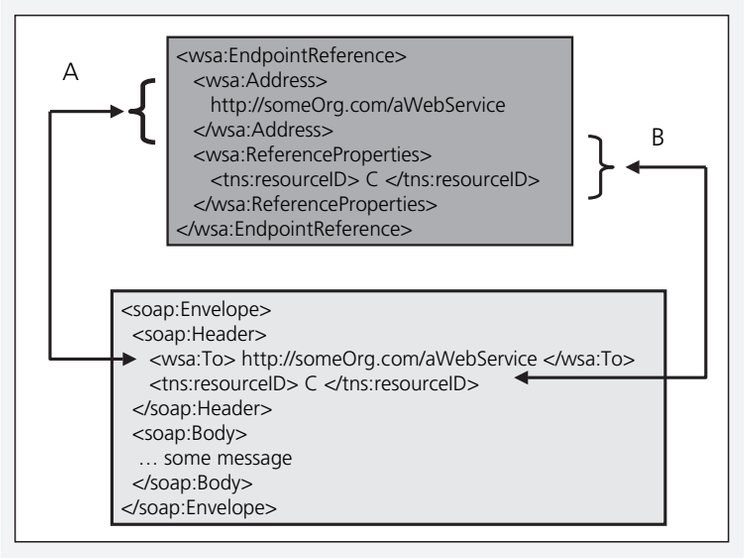

図 -7　エンドポイント・リファレンスと SOAP ヘッダの対応

エンドポイント・リファレンスの SOAPメッセージ・ヘッダへのマッピング

図 -4 の例は，クライアントからの WS- リソースの生成リクエストの結果としてエンドポイント・リファレンスが返るということでしたが，今度は，そうして生成された WS- リソースにクライアントからアクセスする場合を考えましょう．具体的には，Web サービスにおける手続き呼び出しの枠組みを規定している SOAP のヘッダに，**図 -6** のように，すぐ後で見る WS- アドレッシングで規定されたマッピングルールに従って，新しい要素を追加します．

この時は，エンドポイント・リファレンスの XML ドキュメントをそのまま使うのではなく，エンドポイント・リファレンスの情報を，SOAP のメッセージ・ヘッダに次のようなルールでマッピングして利用します（**図 -7**）．ここで重要なことは，エンドポイント・リファレンスの持つ情報と SOAP のメッセージ・ヘッダが持つ情報が，基本的には同じだということです．

- エンドポイント・リファレンスの wsa:Address 情報は，SOAP メッセージの宛先 wsa:To にコピーされる．
- エンドポイント・リファレンスの wsa:ReferenceProperties 情報は，SOAP メッセージのヘッダブロックになる．wsa:ReferenceProperties 内のそ

れぞれの要素が，直接，ヘッダブロックに追加される．

こうして SOAP ヘッダにマップされた，wsa:To 要素と，tns:resourceID 要素が，きちんと WS- リソースの，Web サービス部分と，リソース部分をポイントしているのが分かりますね（図 -7）．

WS-アドレッシングのもう1つの構成要素：「メッセージ情報ヘッダ」

最初に述べたように，WS- アドレッシングは，基本的には，2 つの構成要素からできています．1 つは，これまで見てきた，ネットワーク上でエンドポイントへのポインタを定義しようとする「エンドポイント・リファレンス」です．もう 1 つの構成要素は，HTTP に深く結びついている SOAP メッセージを見直して，メッセージ自体に必要なアドレス情報を埋め込もうという「メッセージ情報ヘッダ」です．

column

REST：HTTP クライアントの「状態」

本文では，HTTP サーバが，基本的には「状態」を持たないという話をしてきました．それでは，HTTP クライアントの「状態」はどう考えればいいのでしょう？

ここでは，WS-RF の「リソース」と「状態」概念に関連して，Roy Fielding の博士論文 [3] で提唱された REST（Representation State Transfer）という，ネットワーク・システムのアーキテクチャ・スタイルの捉え方を簡単に紹介したいと思います．

REST の「リソース」の概念は，WS-RF の「リソース」の概念とは，少し異なっています．REST では，Web ブラウザが，Web 上で URL で一意に指定するものを「リソース」と呼んでいます．REST では，HTTP で定義されている基本的な操作である，POST/GET/PUT/DELETE を，それぞれ，リソースに対する生成／獲得／更新／削除と考えます．これは，WS-RF でのリソース操作よりは，はるかに単純なものです．また，Web の世界は，URL で指定可能な，無数のリソースからなるという考え方は，魅力的で自然な考えだと筆者には思えます．

REST では，リソースそのものとその「表現（Representation）」とを区別します．REST の語源の 1 つになっている Representation（表現）とは，Web のクライアントが，URL で指定してリソースにアクセスした時，Web のサーバが返すもののことです．具体的には，たとえば，Web サーバが返す HTML が，URL で指定されたリソースの「表現」なのです．確かに，Web ブラウザは，HTML の内容を Mime-type に応じて，文字，画像，音声，ビデオ等々にレンダリングしていくわけで，そうして再現されたものがリソースのコピーで，HTML 自体は，その表現にすぎないというわけです．WS-RF でも，後で見るように，WS- リソースは，「リソース･プロパティーズ」という XML ドキュメントでのリソースの「表現」を持ち，リソースはそれを通じてアクセスされます．それと変わりはありません．

Representation State Transfer の State は，何を意味しているのでしょう？ もちろん State は，本論のテーマの 1 つでもある「状態」を意味します．REST では，Web のブラウザが，リソースからその表現を受け取りレンダリングが行われれば，ブラウザは，1 つの「状態」に入ったと見なします．HTML のハイパーリンクをたどって，ブラウザが別の表現を取り込めば，ブラウザは別の「状態」に入ります．

こうして，とりこんだリソースの「表現（Representation）」に従って，クライアントがその「状態（State）」を「転移（Transfer）」してゆくシステムを，REST と呼ぶのです．HTTP クライアントである Web ブラウザは，表現状態遷移マシンということになります．Web サーバは，REST の考え方でも，依然としてステートレスであることに注意してください．

こうした REST の考え方は，現在の Web の世界を抽象化したものです．Web は，REST システムの 1 つの実例なのです．前回のコラムでは，Web/HTTP は，「Eco システムとしての強い拘束力」を持っていると書きましたが，REST は，そうした Web/HTTP の持つ力を，理論的にも技術的にも，正確に再評価しようとする動きだと考えることができます．一見すると，REST の動きは，WS-* での Web サービスの標準化の動きとは独立の動きに見えますが，両者は，深いところでつながりを持っていると，筆者は考えています．

「メッセージ情報ヘッダ」の導入の動機

「メッセージ情報ヘッダ」の導入を動機付けているのは，次のような事情です．

Web サービスの呼び出しの結果としての，サーバからクライアントに向かう HTTP レスポンスのデータをあらためて見てみましょう．そこには，HTTP のヘッダ情報を含めて，このメッセージがどこからどこに向かうのかというアドレス情報はまったく含まれておりません．それには理由があります．HTTP では，クライアントとサーバの間に決まったコネクションが張られていることが前提とされていて，HTTP のリクエストも，それに対してサーバが発した HTTP レスポンスも，この同じコネクションの上を行き来しているからです．

逆に，特定のコネクションが前提されているという想定から離れて，HTTP リクエストや HTTP レスポンスのメッセージを単独でながめれば，これらのメッセージ・データには，自立的なアドレス情報を欠いていることに気付きます．「メッセージ情報ヘッダ」の導入は，その生い立ちからして，HTTP というプロトコルに強く結びついている現在の Web サービスのデータ交換のスタイルを，もう少し一般的に，ネットワーク上のノード間のメッセージ交換のパターンとして整理しようという試みと考えることができます．

メッセージ情報ヘッダの構成

WS- アドレッシングのメッセージ情報ヘッダとは，SOAP のヘッダ部分に，次のような情報を付け加えようというものです．何かメールのヘッダに似ていますね．それもそのはずで，メールのシステムは，最もよく利用されているメッセージ交換システムです．そこでは，メールの「本文」を交換するために，メールのヘッダ情報が利用されています．こうしたメッセージ本体とメッセージ・ヘッダとの関係は，メッセージ交換のシステムにおいては，一般的なものです．

メッセージの宛先を示す <wsa:To> 要素や，そのメッセージがどこから来たかを示す <wsa:From> 要素は，HTTP ではコネクションが先に決まっていると考えれば，HTTP の世界では冗長な情報です．でも，HTTP から見れば冗長な情報を持つことによって，SOAP のメッセージは，HTTP という特定のプロトコルから自由になって，たとえば，SMTP や FTP といった，任意のトランスポート層のプロトコルに対応可能になるのです．

```
---------------------------------------------------------------------
［宛先］（必須）
 <wsa:To> xs:anyURI </wsa:To>
 ［送信元エンドポイント］
<wsa:From> エンドポイント・リファレンス
</wsa:From>
 ［応答先エンドポイント］
<wsa:ReplyTo> エンドポイント・リファレンス
</wsa:ReplyTo>
 ［障害報告エンドポイント］
<wsa:FaultTo> エンドポイント・リファレンス
</wsa:FaultTo>
 ［アクション］（必須）
<wsa:Action> xs:anyURI </wsa:Action>
 ［メッセージ ID］
<wsa:MessageID> xs:anyURI </wsa:MessageID>
 ［関係］
<wsa:RelatesTo RelationshipType="…"> xs:anyURI
</wsa:RelatesTo>
---------------------------------------------------------------------
```

WS- アドレッシングで拡張された SOAP ヘッダのサンプルを次に示します．

```
---------------------------------------------------------------------
<S:Envelope xmlns:S=
“http://www.w3.org/2003/05/ soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/
ws/2004/03/ アドレッシング ">
  <S:Header>
    <wsa:ReplyTo>
      <wsa:Address> http://business456.com/client1
        </wsa:Address> </wsa:ReplyTo>
    <wsa:To> http://fabrikam123.com/Purchasing
    </wsa:To>
    <wsa:Action>
       http://fabrikam123.com/SubmitPO
       </wsa:Action>
  </S:Header>
  <S:Body>
    …
  </S:Body>
 </S:Envelope>
---------------------------------------------------------------------
```

SOAでの，非同期メッセージングのサポート

<wsa:From> 要素や <wsa:RelatesTo> 要素は，行きのメッセージと帰りのメッセージ，あるいは，1 つのメッセージを他のメッセージたちに関連付けます．こうした関連付けは，現在の Web サービスでは標準的な，リクエストとレスポンスが明確なペアになっている，HTTP のようなメッセージ交換のスタイルでは，自明の意味しか持ちません．先に述べたように，こうした関連付けの情報は，HTTP のもとでは，冗長なものです．現在の SOA の主流を成す Web サービスでの，こうしたスタイルのメッセージ交換を同期型のメッセージングと呼ぶことがあります．リクエストを発した側が，レスポンスが返るまで動作をブロックして待ち続け，同期をとるということです．

先に，メッセージ交換システムとしてのメールの話をしましたが，メールはリクエストとレスポンスが常にペアになっているような同期型のシステムではないことに注意してください．10 回メールしてようやく返事が返ってくることもあれば，いくらメールを出しても返事がないこともありえます．メールは，基本的には，一方向のメッセージ（One way Messaging）のシステムです．もらったメールに返事を出すことが可能になっているのは，もらったメールに From ヘッダが含まれているからです．こうした通常のメールのやり取りのパターンは，一方向のメッセージに，それに対する一方向のコールバック・メッセージを非同期に組み合わせた，メッセージ・パターンを形成します．

実は，ビジネスの世界でも，非同期メッセージングに対応するように Web サービスを拡張したいというニーズが存在します．それは，リクエストとレスポンスの間隔が，非常に長い場合でも，Web サービスで対応できるようにしたいというものです．非常に長い時間がかかる処理をどう扱うかというのは，意外と大事な問題なのです．

たくさんのビジネスプロセスが協調しあって，1 つのサービスを提供しているような場合，長く処理時間がかかることがありえます．そのビジネスプロセスの中には，バッチのシステムや，人間の処理が絡む，たとえば，社長の決裁といったプロセスが含まれている場合もあるでしょう．社長の決裁に 1 週間もかかってしまったら，通常の Web サービスの場合には，コネクションがタイムアウトで切れてしまって，レスポンスが返せなくなります．もちろん技術的には，タイムアウトの時間を増やすことは可能なのですが，1 週間ものあいだ，レスポンスが返るまでコネクションを維持するのは，難しいし，リソースの無駄遣いです．リクエスト／レスポンスの枠組みはそのままでも，リクエスト側が張ったコネクションでレスポンスを返すというやりかたをやめて，サーバ側からクライアントへコネクションを張りなおして，レスポンスを返すというアプローチもありえます．こうした拡張に WS- アドレッシングは対応可能です．こうした用途には，このメッセージ情報ヘッダの導入による非同期メッセージングへの対応は，本質的な意味を持つことになります．

SOA の世界では，WS- アドレッシングの導入によって，初めて，同期型のメッセージングから非同期型のメッセージングへの拡張が可能になったのです．

次回予告

今回は，WS- アドレッシングを中心に説明をしてきました．その中で，WS-RF で基本的な役割を果たす，WS- リソースという考え方に触れることができました．次回は，それを受けて，グリッドの世界の標準技術である，WS-RF を紹介したいと思います．同時に，WS-RF の後継技術と目されている，WS-Transfer 関連の技術についても紹介したいと思います．

参考文献

1）Web Service Addressing, http://www.w3.org/Submission/ws-addressing/ (Aug. 2004).
2）WS-ResourceFramework, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
3）Fielding, R. : Architectural Styles and the Design of Network-based Software Architectures, http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (2000).
4）Open Grid Services Infrastructure (OGSI) Version 1.0, http://www.jpgrid.org/information/document/draft-ogsi-service-1.pdf#search=%22OGSI%22

（平成 18 年 9 月 11 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．