連載　Web サービス

第 3 回　ビジネスで使う Web サービス

日本アイ・ビー・エム（株）
天野 富夫　*amano@jp.ibm.com*

編集：XML コンソーシアム

前号では Web サービスが輸出信用状の案内業務の実現に使われている例を紹介しました．Web サービスの普及度合いについてはさまざまな意見がありますが，この事例が示すように，実際のビジネスに使用してそのメリットを享受している企業がすでにいる点はまちがいありません．そうはいっても SOAP で実装を行うことが即ビジネス上のメリットをもたらすとも限りません．今回は企業などがビジネスの中で Web サービスを使う意義について考えてみたいと思います．

◇ ビジネスにとっての Web サービスの価値

輸出信用状の事例では業務の情報を SOAP という一種の封筒に包み HTTP というトランスポートプロトコルを用いて企業間でやりとりしています．多くの人が思い浮かべる典型的な Web サービスのイメージでしょう．ここで採用した SOAP と HTTP は「企業間で業務情報を交換する」というビジネス目標の達成にどのように貢献しているでしょうか．

HTTP を使用することで専用線に比べて初期投資，運用コストともに安価なインターネットを通信インフラとして利用することができます．経済的なメリットの反面，HTTP ／インターネットの（特に企業間での）ビジネス使用には信頼性，可用性，セキュリティなどの点で課題があります．後述するように，これらの課題を解決するための Web サービス関連の技術や規格が提案されています．それらの規格以外に実務的な観点からは業務プロセスの構成を工夫することで問題の影響を最小化することもできます．サーバに蓄積されている信用状を照会するタスクでは何回かリトライを行うロジックを用意することでネットワーク障害に対処することができます．ただし，相手側の状態を更新する（たとえば A さんの口座から 1,000 円引き落とすなど）タスクではそうはいきません．複雑な技術要素を導入しなくても Web サービスのデメリットに対処できるように業務プロセスを構成する（あるいはそのようなプロセスを Web サービス化の対象に選ぶ）ことも Web サービス導入の初期段階における考慮事項でしょう．

次に SOAP 採用のメリットは何でしょうか．筆者は SOAP 仕様によって実現される疎結合だと考えています．ここで疎結合（loose coupling）とは「相互に作用しあうシステム間の依存性が最小であるような構成」[1] をいいます．システム間の依存性は 1 種類ではなく観点の異なるさまざまな依存関係があり得ます．他の分散プログラミング環境と同様，SOAP はシステム間のハードウェア／ OS に関する依存性や実装プログラミング言語に関する依存性を最小化しています．

加えて SOAP 仕様では

- ネットワーク上に分散しているオブジェクト間の依存性
- 使用するトランスポートプロトコルに関する依存性

という 2 つの依存性が最小化されています．前者の依存とは分散オブジェクトの管理（たとえばガベージコレクション）を行うために，各システム上のオブジェクト管理用ミドルウェアが密接に連携する必要があることをいいます．通常，CORBA など同一の規格に沿ったミドルウェア上にアプリケーションを実装することになりシステム間の依存性は高くなります．SOAP 仕様

ではオブジェクト参照をサポートせずに値渡しのみで情報を交換することで分散ガベージコレクションの必要性を排除しています．ちなみにSOAPは発表当初はSimple Object Access Protocolの略称だといわれていましたが，実のところ上記のようなSOAPの設計思想はオブジェクト指向とはあまり関係がありません．SOAP1.2からは何の略かは示さずにSOAPという名前だけが使われるようになりました．

相互に作用しあうシステムのどちらか／または両方が特定のトランスポートプロトコルしかサポートしていないことがあります．SOAPはトランスポートプロトコルから独立して定義されておりオープンであれベンダ固有であれ任意のトランスポートプロトコルと組み合わせて利用することができます．これは適当なアダプタを用意することで相手のトランスポートプロトコルを意識することなく情報の交換が可能になることを意味しています．

このようにSOAPにより広い意味での疎結合が実現可能でありWebサービスが提供するインターオペイラビリティの基礎になっています．Windows系プラットフォーム上のアプリケーションとUnix系プラットフォーム上のそれとを簡単に統合することができるわけです．初期のSOAPやWSDLの仕様には誤りや曖昧な点が存在しベンダ実装間でのインターオペイラビリティを阻害する原因になっていましたが，このこととWebサービスが本質的に持つインターオペイラビリティへの親和性とは別の次元の話です．また，WS-I等の活動によって仕様記述上の問題は解消されつつあります．

本連載の第1回でも触れましたがSOAP採用のデメリットとしてよく挙げられるのは「遅い」，「非効率」という点です．確かにテキストベースのXMLを使い，オブジェクトの中身はすべて値として展開して送るわけですから効率はよくありません．しかし，「速い」「遅い」は何と比べるかによって変わってくる話です．輸出信用状を送るのにコストの問題で専用線は使えないというビジネス状況において，SOAP over HTTPとインターネット上での利用が困難な他の連携方式の速度を比較することに意味があるでしょうか．ビジネス的な観点ではこのまま電話やFAXで処理を続けた場合とのTurn Around Timeや対投資効果が検討されるべきでしょう．確かにSOAPやHTTPを使った実装は他の分散プログラミングの実装と比較して1桁から2桁遅くなることがあります．しかし，人手で1日かかっていた処理がインターネット経由で5分になるとき，さらに専用線が必要な実装を使って1/100に高速化するビジネス上の意義がどれほどあるかはよく考えておく必要があります．

疎結合とはさまざまな観点でのシステム間での依存性に関する概念だと前に述べましたが，「Webサービスは疎結合である」という売り文句にかかわらずSOAPやHTTPを使うだけですべての依存性が最小化されるわけではないことに注意してください．システム間の連携が細粒度の相互作用を頻繁に行うことで実現されている場合両者の依存性は高くなりますが，これはメッセージをSOAPで表現しても変わりません．メッセージの粒度の問題は最近話題のSOA（Service-Oriented Architecture）[2]とも絡んで重要ですが，実装技術だけでなく設計の方法論までを含んだトピックです．

インターネット経由で業務情報を交換する上でSOAPは有力な選択肢ですがさらに簡便な方式をとることもできます．amazon.comでは通常のHTTPのGETやPOSTを使って要求内容を記述しHTTP応答のBody部に結果がXMLで記述されて返ってくる形式のWebサービスを提供しています．amazon.comはSOAPによるWebサービスも提供していますが，使用頻度はREST（Representational State Transfer）Webサービスと呼ばれるこの形式のほうがはるかに多いそうです．Webサービスの利用者であるHP（ホームページ）作成者にとってXSLTを使って応答メッセージからブラウザユーザ（HP作成者にとってのお客様です）向けのHTMLを簡単に生成できるのがその理由です．SOAPに比べると機能や標準化の面で劣る点はありますが，初期の参入障壁を低くできるのは大きなアドバンテージです．

◇ SOAとエンタープライズWebサービス

前章で紹介したREST WebサービスはSOAPを使っていません．要求メッセージにいたっては通常のGETやPOSTそのものでありXMLすら使われていません．SOAP1.2でも同様の呼び出し方が認められています．「Webサービスってを使って呼び出されるのではなかったの？」と疑問に思われる方もいらっしゃるでしょう．

実はWebサービスには前章で述べたような業務情報の交換や連携の実現手段という側面に加えて，システムや業務プロセスを構成するさまざまな機能を一元的に扱うフレームワークという側面があります．前者の観点ではSOAPやHTTPを他の実現手段と比較してその優位性を議論しましたが，後者の観点では比較対象となったライ

バルを含めてすべての手段で実現された機能を Web サービスと見なします．これによって機能を部品として組み合わせて必要なものを作り上げていくという設計が可能になります．

ランタイム時のフォーマットやプロトコルが異なるさまざまな交換・連携手段をみな Web サービスと見なせる秘密は WSDL にあります．WSDL ではメッセージフォーマットの抽象的な定義の記述とランタイムに具体的にどのようなワイヤーフォーマットやプロトコルやトランスポートプロトコルと結びつけるかという記述が分離されています．抽象レベルではすべての Web サービスのインタフェースを同じ枠組みで記述しておき，実際の Web サービスを呼び出すために必要な情報は Web サービス提供者側の個別の実装ごとに指定することができます．WSDL 記述に従って Web サービスを呼び出す WSIF（Web Services Invocation Framework）[3] と呼ばれるオープンソース実装が公開されています．具体的なプロトコルやフォーマットに依存する部分は差し替え可能なプラグインのモジュールとして実装されています．

このような Web サービスの考え方は実装を隠蔽しインタフェースだけを公開した“サービス”を組み合わせることで，既存機能の再利用性を増加しシステムやプロセスの開発や変更を迅速に行おうという SOA とよくマッチします．実は SOA 自体はかなり昔からある概念なのですが，Web サービスという実現のための技術が成熟してきたために最近特に注目されるようになりました．

SOA でサービスを部品として組み合わせるために必須の条件はそのインタフェースが明確に定義されていることです．SOA の観点からは SOAP を情報交換に使うか否かは本質的な条件ではなく状況に応じて選択すればよいものです．実際，機械可読な形式でインタフェースを記述できることが Web サービスの本質である[4] とする考え方があります．「Web サービス＝ SOAP」というイメージが強いため，このような観点での Web サービスをエンタープライズ Web サービスとかエンタープライズサービスと呼んでいます．

SOA においてサービスのインタフェースが重要であることは BPEL4WS（Business Process Execution Language for Web Services）のようなサービスの合成を行う規格を見るとよく分かります．複数の既存サービスを組み合わせて新たなサービスを合成する際，BPEL4WS の記述から各既存サービスの WSDL が参照されています．ある既存サービスからデータをうけとりそれを別の Web サービスへの入力とするといった事柄は WSDL の抽象レベルのメッセージフォーマット記述を参照して指定することができます．BPEL4WS を実行する際には具体的な binding 要素や service 要素の記述が参照されます．適切な WSIF の実装があれば WSDL を用いることで部品として SOAP サービスだけでなく，ローカルな Java プログラム，ベンダ固有のプロトコルで呼び出されるサービス，メインフレームの機能等をサービスとして一元的に扱うことが可能です．

◇ 信頼できる Web サービス

Web サービスをビジネスで使うという文脈でしばしば話題になるのが信頼性の問題です．信頼性にかかわるトピックとしては

- メッセージの送達保証
- （いわゆる）トランザクション処理

などが挙げられます．メッセージの送達保証はメッセージが確実に 1 回だけ相手に届くことを保証する仕組みです．SOAP でこれを行うにはヘッダ部分に制御情報を埋め込んで必要に応じてリトライを行う手順を定義するのが正統的なアプローチです．そのための SOAP の拡張規格が提案されています．一方，世の中にあるトランスポートプロトコルにはすでに送達保証をサポートしているものがあります．SOAP はトランスポートプロトコルとは独立して定義されていますから，そういったプロトコル（たとえば IBM の MQSeries）に乗っかってしまうというのも 1 つの解決手段です．また，相手の内部状態に影響しない照会系の要求や初期化のようなべき等な (idempotent) 要求でしたら応答があるまでリトライするという単純なロジックで事足りる場合もあります．

Web サービスでトランザクション処理を扱う場合に注意すべきことの 1 つはトランザクションの開始から終了までのライフタイムが長い（数日に及ぶこともある）という点です．そのためトランザクションの Atomicity を実現するためにリソースにロックをかけておく手法が使えません．代わりに Compensation（補償）トランザクションという考え方がよく使われます．「A さんの口座から 1,000 円引く」という処理も元に戻すために「A さんの口座に 1,000 円足しこむ」という処理を準備しておくわけです．BPEL4WS では Compensation の対象となる範囲を指定し補償処理を記述する構文が用意されています．トランザクション実行中に例外等が発生した場合，

BPEL4WS 処理エンジンはこの記述に従ってプロセスをトランザクション開始前の状態に戻すことができます．残念ながら補償処理の内容は業務依存であり人間が個別に記述しておく必要があります．Compensation の範囲をどのくらいの粒度で決めたらよいかなど設計上のガイドも整備が必要です．

Web サービスがビジネス粒度のサービスを提供するようになるとそれらで構成されるトランザクションに求められる要件もシステムレベルのトランザクションとは異なってきます．終了の時点でトランザクションの各参加者の状態が一貫したものでなければならないという点は同じですが，一貫した状態とはたとえば「出張のための飛行機の予約，ホテルの予約，レンタカーの予約が終了している」といったビジネスレベルの表現になります．飛行機とホテルの予約に成功すればレンタカーは予約できなくても一貫した状態と見なして予約を保持すべきでしょう．しかし，飛行機の予約に失敗したらホテルやレンタカーの予約はキャンセル（前述の Compensation）したほうがよいかもしれません．ビジネスレベルのトランザクションでは「トランザクション参加者が一貫した状態にあるとはどういうことか」とか「各参加者を一貫した状態にもっていくためにはどうしたらよいか」といった事柄を業務の内容に則して定義しておく必要があるのです．

一般にビジネスレベルのトランザクションを扱うための規格は次の 2 つの要素で構成されています．

- コーディネーション用フレームワーク [5)]
- 個別のコーディネーションプロトコル

コーディネーション用フレームワークはトランザクション参加者が共有するコンテキスト情報の作成や伝達の仕組み，特定のコーディネーションプロトコルへの参加者の登録操作を定義しています．これらの機能自体も Web サービスとして提供されます．コーディネーションプロトコルは各参加者を一貫した状態にもっていくための具体的規約をコーディネーションフレームワークを用いて記述したものです．たとえば WS-Coordination はこのようなフレームワークの一種であり，WS-Transaction はその上で動作するコーディネーションプロトコルを定義しています．

◇ おわりに

Web サービスというと SOAP over HTTP というイメージを持たれる方は多いと思います．しかし，実際ビジネスの場で使おうとすると参入障壁を低くするために SOAP 以外の要求／応答形式が使われたり，信頼性確保のためにベンダ固有のプロトコルと組み合わされたりしています．筆者はこれらすべてを含めて Web サービスだと考えています．ワイヤーフォーマット上ではそれぞれ異なる様相を示すこれらのサービスは設計という観点では「インタフェースを XML で記述できる」という共通した性質を持っています．これこそが Web サービスを Web サービスたらしめる本質的な特徴ではないかと思います．

参考文献
1) Web Services Glossary, http://www.w3c.org/TR/ws-gloss/
2) Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/TP032.pdf
3) Welcome to WSIF: Web Services Invocation Framework, http://ws.apache.org/wsif/
4) Best practices for Web services: Part 1, Back to the basics, http://www-106.ibm.com/developerworks/webservices/library/ws-best1/
5) Web Services Coordination（WS-Coordination）, http://www-106.ibm.com/developerworks/library/ws-coor/

（平成 16 年 8 月 5 日受付）