効率的な多言語間オブジェクト交換手法の設計と実装

古橋貞之[†]　新城　靖[††]　佐藤　聡[††]
中井　央[†††]　板野肯三[††]

1. はじめに

今日では多種多様なサービスがWebを通じて提供されている．これらのサービスには，次に挙げる3つの特徴が多く見られる：

第1に，1つのWebサービスは多数の機能を組み合わせて構築される．例えば写真をWeb上で管理できるようにするWebサービスは，写真をアップロードする機能や，写真を特定の人だけが閲覧できるようにする機能，写真を検索する機能などを組み合わせて構築される．このとき，個々の機能はそれを実装するのに適したプログラミング言語で実装される．例えば，Webアプリケーションの実装にはPerlが利用され，検索エンジンの実装にはC++が利用される．

第2に，Webサービスでは多数のユーザから少しずつ収益を得ることで利益を上げるビジネスモデルが採用されるため，アクセス数に対して得られる利益が小さい．Webシステムにおいて性能の高いシステムを安価に構築するには，1台のサーバをより高性能なものに置き換えるScale-up型の手法より，安価なサーバを多数組み合わせるScale-out型の手法の方が有効であることが知られている[1)]．

第3に，複数のサーバでキャッシュを共有するなどの用途に，単純なバイト列しか保存できない連想配列状のデータモデルを採用した，高速な分散データストアが幅広く用いられている[2)]．これらの分散データストアに構造化されたオブジェクトを格納するには，シリアライズする必要がある．

以上の3つの特徴から，Webサービスを提供するシステムでは，スクリプト言語を含む様々なプログラミング言語で実装されたプログラムの間で，頻繁にデータのやりとりが行われる．1台1台のサーバは安価で性能が低いため，システム全体に占めるシリアライズの負荷の割合が高い．このため，効率的に，すなわち低いCPU負荷と小さいデータサイズでオブジェクトをシリアライズでき，プロセス間で交換できる手法が求められている．

本手法では，まず，効率が良く，スクリプト言語を含む多くの言語で利用できるシリアライズ形式を設計した．次に，そのシリアライズ形式を用いて効率的な遠隔手続き呼び出しを可能にするプロトコルを設計した．

† 筑波大学第三学群情報学類
†† 筑波大学システム情報工学研究科
††† 筑波大学図書館情報メディア研究科

2. 既存手法

HTTP+JSON　WebサービスはWebサーバを用いて構築されるため，オブジェクトをJSON[3)]を用いてシリアライズし，それをHTTPを用いて交換する手法は導入しやすく，使い勝手が良い．しかし，JSONはオブジェクトをすべてテキストに変換してシリアライズするため，オーバーヘッドが大きい．またHTTPは汎用的なテキスト形式のプロトコルであり，数多くの機能拡張がなされているため[4)]，プロトコル解析に要するオーバーヘッドが大きい．

Sun RPC　Sun RPC[5)]はシリアライズの方式としてXDR[6)]を用いており，効率が良い．しかしSun RPCは，Perl，Ruby，Pythonなどのスクリプト言語で簡単に利用することができない．また，サーバやクライアントを実装するために専用のインタフェース記述言語を学習する必要がある．

Protocol Buffers　Protocol Buffersは，Googleが開発し，大規模なシステムで実際的に利用されているシリアライズ形式である[7)]．整数のシリアライズにはBase 128 VariantsとZigZag Encodingと呼ばれる手法が用いられており，小さい整数ほど少ないバイト数でシリアライズする．

3. MessagePack

効率が良く，スクリプト言語を含む多くの言語で簡単に扱うことができるシリアライズ形式を設計・実装し，これを**MessagePack**と命名した．

MessagePackでは，オブジェクトをシリアライズしたデータに，そのオブジェクトが整数であったか配列であったかなどの情報を示す型情報を含めるようにし

た．このためオブジェクトをデシリアライズするために，シリアライズされたデータ以外の情報が何も必要ない．これによりインタフェース記述言語などを用いて型情報を記述することを不要にした．

また，よく使われるデータほど少ないバイト数でシリアライズできるようにした．フラグやエラーコードなどの用途で頻繁に用いられる −32 から 127 までの小さい整数は，型情報とその整数自体を合わせて，1 バイトでシリアライズできるようにした．これらの整数は，Protocol Buffers では倍の 2 バイトになる．また要素の数が 15 個までの小さな構造体や配列，連想配列は，型情報と要素数を示す整数を合わせて，1 バイトでシリアライズできるようにした．さらに，エラーメッセージや小さなデータのやりとりに頻繁に用いられる 31 バイトまでの短い文字列は，型情報とその長さを示す整数を合わせて，1 バイトでシリアライズできるようにした．Protocol Buffers では，少なくとも 2 バイトになる．

また，MessagePack の C および C++向けの実装では，長いバイト列をゼロコピーでシリアライズできるようにした．また C, C++および Ruby 向けの実装では，バイト列をゼロコピーでデシリアライズできるようにした．これにより長大なバイト列でも高速に扱うことを可能にした．Protocol Buffers はゼロコピーではないため，長大なバイト列のシリアライズ・デシリアライズは遅く，負荷も高い．

さらに，C, C++, Ruby, Perl および Python 向けの実装では，オブジェクト 1 つをデシリアライズするためのデータがすべて揃っていない状態でも，デシリアライズ処理を途中まで進められるようにした，この機能を**ストリームデシリアライザ**と呼ぶ．これによって，MessagePack でシリアライズされたオブジェクトを TCP ソケットやパイプを通じて受信する際に，後に続くデータを待ち受けている間にデシリアライズ処理を途中まで進めておくことができる．これによりデシリアライズ処理に要する遅延を隠蔽することを可能にした．

現在 MessagePack の C, C++, Ruby, Perl および Python 向けの実装をオープンソースソフトウェアとして公開している[8]．このため，Web サービスで即時利用することができる．

4. MessagePack-RPC

MessagePack を利用して効率的な遠隔手続き呼び出しを可能にするプロトコルを設計・実装し，これを**MessagePack-RPC** と命名した．

MessagePack-RPC では，Request, Response, Notify の 3 種類のメッセージを MessagePack を用いて交換する．Request は，[0, メッセージ ID, メソッド番号またはメソッド名, 引数] の 4 つの要素からなる配列である．Request を受け取ったホストは，Request と同じメッセージ ID を含んだ Response を返送することが期待される．Response は，[1, メッセージ ID, エラー情報, 結果] の 4 つの要素からなる配列である．Notify は，[2, メソッド番号, 引数] の 3 つの要素からなる配列である．Notify は Request と似ているが，Response を返送する必要がない．

MessagePack-RPC の Ruby 向けの実装をオープンソースソフトウェアとして公開している[9]．Ruby 向けの実装はインタフェース記述言語を学習することなく利用でき，非常に少ないステップ数でクライアントやサーバを実装できるようにした．

5. おわりに

本稿ではまず，スクリプト言語を含む多くの言語で効率的に利用できる通信手法の重要性について述べた．次に，効率が良く，多くの言語で利用できるシリアライズ形式である MessagePack と，それを用いたオブジェクト交換手法である MessagePack-RPC の特徴について述べた．今後は詳細な性能評価を行う．

参考文献

1) Michael, M., Moreira, J. E., Shiloach, D. and Wisniewski, R. W.: Scale-up x Scale-out: A Case Study using Nutch/Lucene, *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8 (2007).
2) Fitzpatrick, B.: Distributed caching with memcached, *Linux J.*, Vol. 2004, No. 124, p. 5 (2004).
3) Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (2006).
4) Nottingham, M. and Mogul, J.: HTTP Header Field Registrations, RFC 4229 (2005).
5) Thurlow, R.: RPC: Remote Procedure Call Protocol Specification Version 2, RFC 5531 (2009).
6) Eisler, M.: XDR: External Data Representation Standard, RFC 4506 (2006).
7) Pike, R., Dorward, S., Griesemer, R. and Quinlan, S.: Interpreting the Data: Parallel Analysis with Sawzall, *Scientific Programming Journal*, Vol. 13, pp. 277–298 (2005).
8) Sadayuki Furuhashi: MessagePack (2009). `http://msgpack.sourceforge.jp/`.
9) Sadayuki Furuhashi: MessagePack for Ruby (2009). `http://rubyforge.org/projects/msgpack/`.