Vol.2016-IOT-35 No.2
Vol.2016-SPT-20 No.2
2016/9/23

RESTful APIのためのソースコード自動生成ツールの設計

上田 達也[1,a)]　松本 早紀[1,b)]　灰野 有香[1,2,c)]

概要：IT サービスの Web 化の進展と，サービスのライフサイクルの短期間化に伴い，開発生産性の向上が求められている．方策のひとつしてサービスの View と Model を分離する目的での Web サービスの API 化が挙げられるが，この方式では，開発フェーズでの通信仕様の変更が工数や工期の延長というコスト増の要因のみならず，ソフトウェアの品質の低下をも招く場合がある．
そこで著者らは，ソースコード自動生成フレームワークである blanco Framework を利用して，RESTful API のための通信部スタブコードを自動生成するツールを開発した．本ツールでは Excel シート上に定義した API 仕様書を読み取り，PHP や Java など複数のプログラミング言語のためのスタブコードを生成する．Web アプリケーションは，クライアントサイド，サーバサイドがそれぞれにこのスタブコードを利用して通信部を実装することで，クライアントサーバ間での通信仕様の整合性を担保することを目的としている．

キーワード：ソースコード自動生成, RESTful API

A design of a source code generator for RESTful API system

Tatsuya Ueda[1,a)]　Saki Matsumoto[1,b)]　Yuka Haino[1,2,c)]

Abstract: As evolving of Web services and reducing of lifecycles of the services, it is required to increase the development productivity. As one of the measures to this, Application Programming Interface (API) are often provided as server side functions of Web services to separate views from models, however, on this method, it is difficult to change specifications of communication protocols on development phase of projects, because it may cause increasing of cost and period of the projects.
So we design and develop an automated source code generator which generates stub-code for RESTful API communications by using blanco Framework , which is source code generation framework. This tool reads the API specifications defined on Excel worksheets and generates stub-codes (abstarct codes) for multiple programming languages like PHP, Java and so on. By using these stub-codes, both server and client side of Web applications can implement services with consitency of communication protocols.

1. はじめに

近年，インターネット上の公開サービスだけでなく，クローズドな業務系システムでも Web サービスの手法を用いたシステム開発が多くおこなわれている．Web サービスの利点は，1）ユーザインタフェイスとして Web ブラウザを利用するので OS や端末ごとに個別開発をする必要がない，2）結果として開発工数，工期が短縮され企画から短期間でサービスリリースまで持ち込める，などが挙げられる．

また高度な情報化社会といえる現代社会では，ユーザのニーズの変遷も速い．特にユーザ体験の嗜好の変化は速く，サービス自身のライフサイクルもますます短くなっている．このような状況では，ソフトウェア開発においてより一層の生産性の向上が求められている [1]．

オブジェクト指向プログラミングにおける生産性の向上

[1] 有限会社うえだうえおうぇあ
Ueda Ueo Ware, Inc.
[2] 大阪市立大学創造都市研究科
Graduate School for Creative Cities, Osaka City University
[a)] ueo@ueo.co.jp
[b)] kinoko@ueo.co.jp
[c)] haino@ueo.co.jp

という観点からは，遅くとも 1988 年には，MVC パターンが Krasner らによって提唱され [2]，近年ではユーザインタフェイスをもつシステムでは MVC パターンを用いた設計をおこなうことが，分野を問わず推奨されている．特に Web アプリケーションの分野では，サーバサイドではアプリケーションの機能のみを Application Programming Interface（**API**）として提供し，クライアントサイドは単独で動作するアプリケーション（Web ブラウザに依存するものを含む）として実装される方式が採用される例が増えている．

このように model と view の結合度をさらに疎にすることで，たとえばユーザの嗜好の変化に対するための（機能変更を伴わない）デザイン変更などはサーバサイドの改修をすることなくユーザ体験を向上することができる．これはサーバサイドの機能テストを省略できることを意味し，結果として，運用フェーズまで含めた生産性の向上が期待できる．

また開発チームの編成の点でも，サーバサイド開発チームとクライアントサイド開発チームを別個に編成し，地理的・時間的な制約を最小限に抑えた開発体制を構築することが容易となる．

ただしこの方式では，通信仕様の変更があった場合にはサーバ・クライアントの両サイドの改修が必要となる上に，両チーム間で変更点の認識の整合性が間違いなく取れる様にすることや，変更のタイミングの意識あわせなど，プロジェクト管理の面でのコストが発生することに注意が必要である．

最近ではさらに，システムの各機能を比較的小規模な Web サービスに分割し，各機能間を API で連携することで大規模なシステムを構築する「マイクロサービスアーキテクチャ」[3] と呼ばれる手法も採用されつつある．マイクロサービスアーキテクチャでは，システム内で多数かつ多岐にわたる API を定義し実装することになる．また各マイクロサービスは API を提供するサーバでありつつ，他のサービスの API を利用するクライアントにもなる．このような状況では，通常の開発手法では，API の一貫性を保つために相当の管理工数がかかる事が予想される．

API 化された Web アプリケーションを開発する際のこれらの課題を解決するために，著者らはソースコード自動生成フレームワークである blanco Framework を利用して，RESTful API のための通信部スタブコードを自動生成するツール（**blancoRest**）を開発した．

本稿では，第 2 章で Web アプリケーションの API 化について，第 3 章で RESTful API の概要を，第 4 章で blanco Framework の概要を述べる．第 5 章で著者らが開発した blancoRest の設計について具体的に述べ，第 6 章で結論と今後の課題について述べる．

2. Web サービス

近年，IT サービスを Web アプリケーションとして構築することが広く行われている．アーキテクチャの異なるクライアントでも，Web ブラウザの描画企画が共通であればクライアントアプリの移植が不要であることが普及の一因と考えられる．当初はユーザインタフェイスの簡略化が主な目的であったが，HTML5 や CSS の普及によって，より表現力の高い複雑なアプリケーションが作られるようになっている．

さらに最近では，開発生産性の向上と保守性の向上を目的として，Web アプリケーションの view と model を完全に分離し，model を API として提供することも多い．

Web アプリケーションでは通常，通信プロトコルには HTTP[*1] が用いられる．このため API 化された Web アプリケーションを構成する要素は URL，ヘッダおよびペイロードとなる．

個々の API は URL で表現され，API のメタ情報はヘッダに乗せられる．API の本体たるリクエスト情報（パラメータ等）とレスポンス情報（戻り値等）は通常ペイロードに乗せられるが，これらの情報を構造化する書式としては Extensible Markup Language（**XML**）[4] または JavaScript Object Notation（**JSON**）[*2] が用いられる場合が多い．

構造化形式として XML が採用されているものとしては，Simple Object Access Protocol （**SOAP**）[5] が挙げられる．SOAP 自身はデータの運搬手段を想定してないが，HTTP を介して SOAP 通信を行うものを，W3C は **Web サービス** [6] と呼ぶ．Web サービスでは API の定義情報も XML で定義することでき（Web Service Description Langage = WSDL），処理系に依存しないサービスを構築するが可能となっている．

最近では，RESTful API（第 3 章）と呼ばれる方式が採用されることもある．RESTful API も，ペイロードに乗せるデータ形式としては XML が用いられる場合もあるが，JSON が用いられることが多い．XML，JSON ともにテキストデータであるため，HTTP のペイロードとしても，また人間がデータを直接読み取る場合にもデータの取り扱いが容易なことなどの利点は共通であるが，同じ情報を表現する場合には JSON の方がより小さなデータサイズで表現できることによると思われる．

本稿では，W3C が定義する SOAP 通信のものに加えて，RESTful API による Web アプリケーションなど，HTTP をキャリアとして API を提供する IT サービスを「広義の Web サービス」と呼ぶことにし，単に Web サービスと称した場合は広義の Web サービスを指す．W3C が規定

[*1] rfc7230-7235
[*2] rfc7159

する SOAP 通信を前提とした Web サービスを指す場合は「SOAP ベースの Web サービス」と呼ぶ．

3. RESTful API

RESTful API は，Fielding によって提唱された Representational State Transfer (**REST**) アーキテクチャ [7] の原則に基づいて設計された Web サービス API の総称，あるいは設計思想である．

Fielding は参考文献 [7] で，REST の特徴としていくつかの要素を列挙している．なかでも 1 ）クライアント・サーバ，2 ）ステートレス，3 ）統一されたインタフェイスといった特徴が，REST アーキテクチャの原則に基づいた Web サービス実装が備えるべきものと解釈されることが多い．

RESTful API では，統一されたインタフェイスとして Uniform Resource Identifier （URI）を用い，サービスをネットワーク上のリソースとして表現する．リソースに対して HTTP Method（GET,POST,PUT,DELETE）の意味に基づいて統一的な処理をすることで，REST アーキテクチャの原則を実現する．

リソースの操作に際しては，HTTP Method に加えて，データや条件を指定する場合がある．これらのデータや条件などは XML または JSON 形式で与えられ，HTTP パラメータあるいは HTTP Request のペイロードに乗せられる．またリソース操作の結果は HTTP Response のペイロードとして戻される．

最近では多くは JSON が用いられる傾向があるが，XML に比べて通信量が削減できること，多くの Web ブラウザで動作する JavaScript を使用したクライアントとの親和性が高いことなどが理由だと考えられる．

RESTful API は SOAP ベースの Web サービスよりも後発ではあるが，facebook や google，twitter など大手のインターネット上のサービスが，公開 API として RESTful API を採用していることから SOAP ベースのものより普及していると言える．そのシンプルかつ統一された操作性が，普及が進んだ要因のひとつと考えられる．

4. blanco Framework

ソースコード自動生成ツールである blanco Framework[8] の主な特徴は以下の通りである．

- データ構造やオブジェクトの定義を Excel シートで行い，ソースコードを自動生成する
- 多数のプログラミング言語に対応している
- ソースコードは各言語の文法に則って（テンプレート方式ではなく）スクラッチ方式で組み立てられる

著者らが RESTful API のソースコード自動生成ツールとして blanco Framework を採用した主な理由は 1 ）Excel シートによる構造定義，2 ）多数のプログラミング言語に対応している，の 2 点による．

RESTful API のデータ構造を自動生成するツールは他にもあるが，多くが，データ構造やオブジェクトの定義に JSON あるいは YAML[9] を採用している．特に YAML は，ポータブルなオブジェクト定義書式として普及しているが，プログラミングや情報系の専門教育を受けていない人にとって容易に理解できるとは言いがたい．

近年，少人数の技術者あるいは技術に精通したマネージャが集まってインターネット上で新しいサービスをスタートする「スタートアップ」と呼ばれる事業形態が増えている．このような場合には表現力に利点のある YAML や実際に通信で用いる JSON でのデータ構造の定義は開発生産性の向上に大いに寄与するだろう．

しかし，一般企業における業務システムの構築に際しては（Web サービスの導入は進んではいるが），発注者側の担当者が必ずしも技術者あるいは技術に精通しているとは限らない．この様な場合，YAML 等の定義はソースコードと同様の扱いをうけ，ドキュメントとしての納品は受け入れられない場合が多い．

YAML から HTML などのより可読性の高い形式への変換は可能であるが，技術資料といえども Excel 等一般的な Office ソフトで印刷に適した形に整形した上での納品を求められることも多い．HTML 形式や自動生成された Excel 表形式の場合，印刷可能なサイズへの調整など，人力での修正が必要となり，コストの増大と作業ミスによるドキュメントの精度の低下，さらには納品ドキュメントと実装との解離が懸念される．

blanco Framework では整形済みの Excel シートを定義書として用いるため，定義書をそのまま納品物として使用できる．定義に変更があった場合には定義書を修正してソースコードを再生成するので，納品ドキュメントと実装との解離が生じない．

逆に公開などの目的で HTML 形式でのドキュメントが必要な場合も，Excel シート定義書の「説明」欄に記述した内容がソースコードに Javadoc 形式で反映されるので，生成物から自動生成する事が可能である．

また，Web サービスではサーバとクライアントが疎結合であることが重要であり，サーバとクライアントで異なる処理系，プログラミング言語で開発されることも許容している．このため，生成されるソースコードは多言語に対応していなければならない．

これらの理由から，本システムでは blanco Framework を採用することとした．

5. blancoRest の設計

本章では，blanco Framework を利用した RESTful API のための通信部スタブコード自動生成ツールの設計について述べる．本ツールは blanco Framework に属する一ツー

ルとして実装するため，**blancoRest** と命名する．

5.1 Web サービス実装時の課題

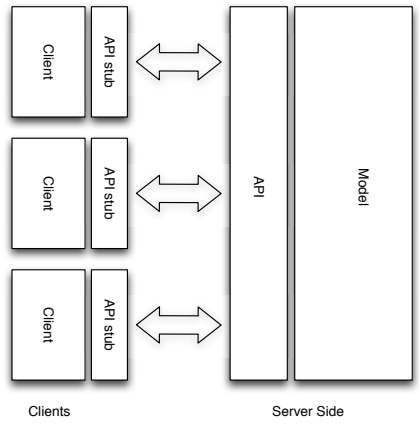

図 1　Web サービス API 概念図

API 化された Web サービスでは，サーバサイドにサービスの主機能が配置され，クライアント側からサーバ側の API を呼び出すことでサービスを提供する（図 1）．

このようなシステムを開発する場合，一般的には以下の手順がとられる．

- API 仕様の定義
- サーバ側通信部の構築
- クライアント側通信部の構築
- 疎通確認
- model の実装（サーバ側）
- view,control の実装（クライアント側）

このとき，考えられる課題のいくつかを列挙する．

5.1.1 複数のクライアントプラットフォーム

クライアントプラットフォームは複数存在しうる．プラットフォーム毎に使用するプログラミング言語が異なることも多い．すなわち API 仕様書は言語やプラットフォームに依存しない形で定義する必要がある．

またクライアント側通信部もプラットフォーム毎に個別に実装する必要がある．実装工数の増大もさることながら，使用言語が異なる場合には，API 仕様書に則った実装になっているかの検証にかかる工数も相当な規模になると考えられる．

5.1.2 通信仕様の変更

システム開発においては，内的あるいは外的要因によって仕様変更を余儀なくされることはよくある．通信仕様のようなシステムの根幹をなす仕様はなるべく変更されるべきではないが，設計段階にすべての要因を考慮して完全な設計を行うことも難しい．

このように通信仕様が変更されると，5.1.1 節で挙げた課題による工数面での負担が都度発生することになり，ひいては開発品質の低下にも繋がりかねない．

5.1.3 API 通信部の自動生成

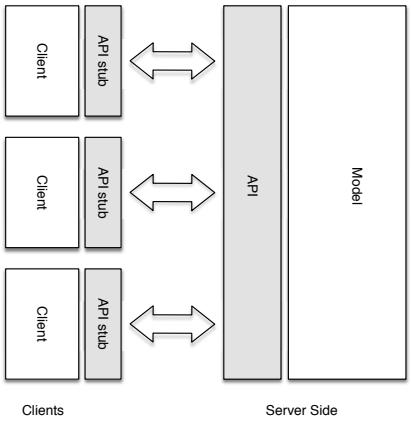

図 2　API 通信部の自動生成

これらの課題の解決方法として，サーバ・クライアント両側の通信部分のコードを仕様書から直接自動生成できると効果が高い．図 2 で影をつけた部分を共通の仕様書から各プラットフォームで使用する言語向けに自動生成することで，1）コード記述の手間が省ける，2）コーダによる品質の差異が生じないといったメリットが生じる．

加えて仕様変更に対しても耐性が高い．これは単にコード記述量が減少するためだけでなく，コードの生成元となる仕様書が適切に共有されてさえいれば，すべてのプラットフォームで仕様変更の反映漏れがなくなるからである．

仕様書の共有については，バージョン管理システムの活用によって技術的に担保できる．

5.2 自動生成の範囲

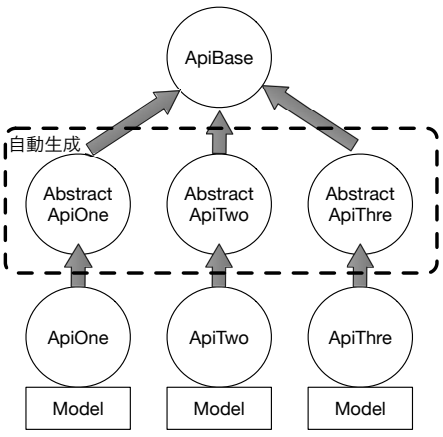

図 3　電文処理における自動生成の範囲

本ツールでは，クライアントからの呼び出しによってリクエストを受け付けレスポンスを返す処理を**電文処理**と呼ぶ．またリクエストやレスポンスを表すデータ構造（オブ

ジェクト）を**電文**と呼ぶ．

電文処理も電文も Excel シートで定義され，電文は blanco Framework の機能によって対象言語でのオブジェクト定義として自動生成される．電文処理は図 3 のように構成され，このうち破線囲みで示した抽象クラス部分が自動生成の対象となる．

電文処理の共通処理は ApiBase と呼ばれる抽象クラス上に実装する．これには JSON と処理系でのオブジェクト形式間の相互変換や，値やオブジェクトの正当性検査（バリデーション）などが含まれる．

自動生成される抽象クラスは blancoRest によって統一された書式で生成されるので，ApiBase に記述する処理コードはシステムにあわせた形でいちど書いてしまえば変更の必要はない．従って ApiBase を自動生成する必要はない．

また電文処理の本体は自動生成された抽象クラスを拡張して作成する．こちらは API ごとに独自の実装をするので自動生成の対象外である．

すなわち自動生成の範囲は，共通処理を実装するベースクラスと各 API の処理を実装するクラスとをつなぎ，各 API 実装に共通処理を提供する部分になる．これによって blancoRest ではサーバ・クライアント間で RESTful API におけるリクエストとレスポンスの一貫性が担保される．

5.3 型の指定

処理系に厳密な型が存在し，コンパイル時に型チェックが行われると，仕様変更による抽象クラスの修正が発生した場合に，各 API の実装クラスでコンパイルエラーが発生する．コーディング作業を担当するプラグラマはコンパイルエラーの発生を嫌うが，プロジェクト全体としてシステムの品質を担保する要素として，厳密な型チェックによるエラーの発生は好ましい．

なぜなら，コンパイル時のエラーは結合テストや本番稼働時に発生するエラーよりも影響範囲が狭いからである．厳密な型チェックが無い場合本番稼働するまで型の不整合が露呈しなかったり，さらに悪いことには，ごく希にしか成立しない特定の条件でのみエラーが発生し，本番稼働から相当な時間を経てからシステムが停止してしまうこともありうる．

blancoRest では，定義書の変更によって各 API の実装クラスでコンパイルエラーが発生することで，システム全体の品質を向上することと，結果として開発生産性が向上することを期待して，5.2 章で示したクラス構成をとっている．

しかし実際には PHP や Javascript など，型チェックが厳密でない（あるいはオブジェクトが動的定義できる）処理系が使用されることも多い．この場合，残念ながら blancoRest の型チェック機構は充分には機能しないので，単体テストの充実な別の手法で品質の維持を図る必要がある．

6. おわりに

本稿ではまず Web アプリケーションの概念を再確認し，REST アーキテクチャの原則に基づいた Web サービスを構築する際の開発生産性をあげる方式について考察をした．その結果 API 通信部ソースコードの自動生成が効果があると考え，blancoRest を開発し，その設計について記述した．

blancoRest の効果を最大限に活かすためには，処理系が厳密な型チェックをする必要があるが，サーバサイドでシェアの高い PHP や，Web アプリケーションのクライアントとしては無視できない JavaScript では型チェックが厳密にはできない点にどう対処するかが，今後の課題となる．とはいえ，サーバサイドでは Java などの型チェックが厳密な処理系を採用することは比較的容易であるが，クライアントサイドで JavaScript を否定することは現実的では無い．

そこで現在，クライアントサイドに TypeScript[10] を採用することを検討している．TypeScript は JavaScript に対して上位互換であり，配備前に TypeScript から JavaScript への変換作業が必要なため，現時点で型チェックが行われ，不整合の場合にはエラーが発生する．

ただし blanco Framework は現時点で TypeScript に対応していないので，まず blanco Framework を TypeScript に対応する作業を予定している．

参考文献

[1] 細野　繁，赤坂文弥，木見田康治，下村芳樹：クラウド環境における Web サービスの設計とライフサイクル管理，日本機械学会論文集（C 編），Vol. 79, No. 808 (2013).
[2] Krasner, G. and Pope, S.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80, *J. Object Oriented Program*, Vol. 1, No. 3, pp. 26–49 (1988).
[3] Fowler, M. and Yves Lafon, W.: Microservices, a definition of this new architectural term, http://martinfowler.com/ (online), available from ⟨http://martinfowler.com/articles/microservices.html⟩ (accessed 2016-08-25).
[4] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F. and Cowan, J.: Extensible Markup Language (XML) 1.1 (Second Edition), The World Wide Web Consortium (W3C) (online), available from ⟨https://www.w3.org/TR/xml11/⟩ (accessed 2016-08-29).
[5] Mitra, N. and Lewis, J.: SOAP Version 1.2 Part 0: Primer (Second Edition), The World Wide Web Consortium (W3C) (online), available from ⟨https://www.w3.org/TR/soap12-part0/⟩ (accessed 2016-08-25).
[6] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D.: Web Services Architecture, The World Wide Web Consortium (W3C) (online), available from

⟨https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/⟩ (accessed 2016-08-29).
[7] Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures, *Doctoral dissertation, University of California, Irvine* (2000).
[8] 伊賀敏樹：blanco Framework, (online), available from ⟨https://osdn.jp/projects/blancofw/⟩ (accessed 2016-08-29).
[9] Ben-Kiki, O., Evans, C. and dt Net, I.: YAML: YAML Ain't Markup Language, (online), available from ⟨http://yaml.org/⟩ (accessed 2016-08-29).
[10] : TypeScript, Microsoft (online), available from ⟨http://www.typescriptlang.org/⟩ (accessed 2016-08-31).

付　　録

A.1 API 定義書

電文定義書　　様式 ver 0.8.0 (2016.04.08)

1. この電文定義書は、blancoSOAPが入力ファイルとして利用します。
2. 定義書様式に記入された情報から、xsdファイルやValueObjectとして利用するJavaソースコードなどが自動生成されます。

電文定義・共通	
電文ID	ApiSampleGetRequest
説明	サンプルAPIの要求電文です。
電文種類	要求電文(C→S)
電文のMETHOD	GET
第1パラメータ	
第2パラメータ	
第3パラメータ	
第4パラメータ	
許可（所有者）	
許可（グループ）	
許可（その他）	
電文の親クラス	ApiGetTelegram
名前空間	BlancoRestPhp
パッケージ	api.sample

電文定義・一覧

No.	項目名	項目の説明 JavaDocに利用されます	型	総称型	必須	デフォルト	長さ Min長	長さ Max長	値範囲Check Min値	値範囲Check Max値	形式Check 正規表現	備考
1	field_1	フィールド1	string		YES		0	10				
2	field_2	フィールド2	integer						0	100		
3	field_3	フィールド3	boolean			TRUE						
4	field_4	フィールド4	float									
5	field_5	フィールド5	array	ObjectSample								array<ObjectSample>
6	field_6	フィールド6	object									
7	object_sample	フィールド7	ObjectSample									

図 **A·1**　定義書 (request) の例

バリューオブジェクト定義書(php)　　様式 ver 0.0.1 (2006.11.20)

1. このバリューオブジェクト定義書は、blancoValueObjectPhpが入力ファイルとして利用します。
2. 定義書様式に記入された情報から、バリューオブジェクトのためのPHPソースコードが自動生成されます。

バリューオブジェクト定義(php)・共通	
クラス名	ObjectSample
パッケージ	api.sample.valueobject
説明	バリューオブジェクトのサンプル。このクラスは単にサンプルです。実際の動作には利用されません。
ファイル説明	バリューオブジェクトのサンプル。このクラスは単にサンプルです。実際の動作には利用されません。

バリューオブジェクト定義(php)・継承	
クラス名	

バリューオブジェクト定義(php)・一覧

No.	フィールド名	型	総称型	デフォルト	説明
1	stringField1	string		デフォルト	デフォルト値あり
2	stringField2	string			デフォルト値なし
3	booleanField1	boolean		false	デフォルト値あり
4	booleanField2	boolean			デフォルト値なし
5	intField1	integer		123	デフォルト値あり
6	intField2	integer			デフォルト値なし
7	arrayField	array			

図 **A·2**　定義書 (valueObject) の例