情報処理学会論文誌 Vol.55 No.11 2451–2460 (Nov. 2014)

mod_mruby：スクリプト言語で高速かつ省メモリに拡張可能なWebサーバの機能拡張支援機構

松本 亮介[1,a)] 岡部 寿男[2]

受付日 2014年2月20日, 採録日 2014年9月12日

概要：Webサービスの大規模・複雑化にともない，Webアプリケーションの開発だけでなく，Webサーバソフトウェアの機能拡張も必要になる場合が多い．Webサーバの機能拡張において，高速かつ軽量に動作することを重視した場合，C言語による実装が主流であったが，生産性や保守性を考慮した場合はスクリプト言語で機能拡張を行う手法も提供されている．しかし，従来手法は，Webアプリケーションの実装だけでなく，Webサーバの内部処理を拡張することを主目的とした場合，高速性・省メモリ・安全性の面で課題が残る．そこで，スクリプト言語で安全に機能拡張でき，かつ，高速・省メモリに動作するWebサーバの機能拡張支援機構を提案する．Webサーバプロセスから内部処理としてスクリプトが呼び出された際，高速に処理するために，インタプリタの状態を保存する状態遷移保存領域の生成を，サーバプロセス起動時に生成しておいて，それを複数のスクリプトで共有して実行するアーキテクチャをとった．また，メモリ増加量を低減し，かつ，状態遷移保存領域を共有することにより生じるスクリプト間の干渉を防止して安全に機能拡張するために，スクリプト実行後に状態遷移保存領域からメモリ増加の原因となるバイトコード，および，任意のグローバル変数・例外フラグを解放するようにした．このアーキテクチャの実装には，組み込みスクリプト言語mrubyとApacheを利用し，Rubyスクリプトによって容易にApache内部の機能拡張を行えるようにした．このApacheの機能拡張支援機構をmod_mrubyと呼ぶことにする．

キーワード：Web server，Apache，mruby，mod_mruby，interpreter

mod_mruby: A Fast and Memory-Efficient Web Server Extension Mechanism Using Scripting Language

Ryosuke Matsumoto[1,a)] Yasuo Okabe[2]

Received: February 20, 2014, Accepted: September 12, 2014

Abstract: As the increase of large-scale and complex Web services, not only a development of Web applications but also an implementation of Web server extensions is required in many cases. The Web server extensions were mainly implemented in C language because of fast and memory-efficient behavior, and extension methods using scripting language also are provided with consideration of maintainability and productivity. However, if the existing methods primarily intended to enhance not the implementation of Web applications but the implementation of internal processing of the Web server, the problem remains in terms of fast, memory-efficiency and safety. Therefore, we propose a fast and memory-efficient Web server extension mechanism using scripting language. We design the architecture that a server process create the region to save the state of the interpreter at the server process startup, and multiple scripts share the region in order to process fast when the script is called as internal processing from a Web server process. The server process free any global variables, the exception flag and the byte-code which cause the increase of memory usage mainly, in order to reduce the memory usage and extend safely by preventing interference between each scripts because of sharing the region. We implement the mechanism that can extend the internal processing of Apache easily by Ruby scripts using Apache and embeddable scripting language mruby. It's called "mod_mruby".

Keywords: Web server, Apache, mruby, mod_mruby, interpreter

1. はじめに

大規模な Web サービスの普及や Web ホスティングサービス [1] の低価格化にともない，企業だけでなく個人も Web サイトや Web サービスを持つ時代になってきている．さらに，スマートフォンの普及により，インターネットはより身近なものになってきている．その結果，Web サーバへのアクセス数は日々増加してきており，Web サービス事業者は，Web サービスの大規模・複雑化にともない，いかにセキュリティを担保しながらも安定してアクセスを処理できるか，いかにサービスの運用・管理コストを低減できるかが課題となっている．さらに，Web サービスを高品質かつ短納期に実装する必要があり，保守性も意識しなければならない．

Web サービスを安定して提供するために，Web サーバソフトウェアそのものの内部機能の拡張が必要となる場合が多い．そのような状況で，Web サーバソフトウェアの拡張を実装しやすくするためのフレームワークも提案されている [2]．Web サーバソフトウェアの内部機能を拡張することにより，Web コンテンツ処理前に，リソース制御やアクセス制御等の緻密な制御を実現できる [3], [4], [5]．これまで，Web サーバの拡張は，高速かつ軽量に動作することを重視して C 言語による実装が主流であったが，生産性や保守性を顧慮して，スクリプト言語で機能拡張を行う手法が提供されている [6], [7], [8]．しかし，従来手法の mod_perl [6] や mod_ruby [7] は，高速に処理するために，インタプリタを複数のスクリプトで共有するが，アプリケーションとしてまったく別々のスクリプト間でもグローバル変数名が干渉し合い，拡張機能を安全に実装できない．また，Perl や Ruby のインタプリタやライブラリが巨大であるため，高速かつ省メモリに Web サーバそのものの内部処理を拡張するには適していない．一方，従来手法の mod_lua [8] は，Lua [9] が高速かつ軽量な組み込みスクリプト言語であることを利用して，スクリプト実行ごとにインタプリタを用意することにより安全に実装できるようにしている．しかし，スクリプト実行ごとにインタプリタの読み込み・解放とライブラリの読み込みが必要となり，C 言語による実装に比べれば依然として処理効率が悪い．

我々は，高速かつ軽量に動作する組み込みスクリプト言語に着目し，インタプリタを Web サーバプロセスに最適化して組み込むアーキテクチャを設計することにより，スクリプト言語で安全に機能拡張でき，かつ，高速・省メモリに動作する Web サーバの機能拡張支援機構を提案する．Web サーバプロセスから内部処理としてスクリプトが呼び出された際，高速に処理するために，インタプリタやスクリプトに関する情報を保存しておく領域（以降，状態遷移保存領域とする）を，サーバプロセス起動時のみに生成しておいて，それを複数のスクリプトで共有し実行するアーキテクチャをとった．また，メモリ増加量を低減するために，スクリプト実行後に状態遷移保存領域から，メモリ増加の大きな原因となるバイトコードのみを解放するようにした．さらに，状態遷移保存領域を共有することにより生じるスクリプト間のグローバル変数や例外処理の干渉を防止するために，スクリプト実行後は，状態遷移保存領域から任意のグローバル変数・例外フラグを解放するようにした．このアーキテクチャにより，稼働し続けることが前提のサーバプロセスに対し，高速性・省メモリ・安全性の観点から最適化してインタプリタを組み込み，スクリプト言語でも効率良く内部機能拡張を実装できる．

実装は，Apache HTTP Server [10]（以降 Apache とする）に組み込みスクリプト言語 mruby [11] を組み込むことにより，Apache の内部機能を，Ruby スクリプトによって容易に拡張できるようにした．この Apache の機能拡張支援機構を mod_mruby と名付けた．mod_mruby は，Ruby スクリプトで内部機能を拡張できるように，C 言語と連携が容易な組み込みスクリプト言語 mruby の特性を生かして，Apache の内部処理やデータと連携するための API を実装した．mod_mruby の実装に必要な改良として，mruby に対しても，バイトコードを解放するための組み込み API を標準機能として実装したり，mruby の組み込みを容易にする機能を追加したりした．

mod_mruby では，Ruby スクリプトを変更することで，即時に Apache の内部機能が変更・拡張可能となり，コンパイルや Apache の再起動を必要としない．また，さらに高速に処理したい場合は，つど Ruby スクリプトを呼び出さずに，サーバ起動時にバイトコードまではコンパイルしておいて，処理を呼び出されたときにバイトコードから実行することも可能である．バイトコードから実行するアーキテクチャにより，C 言語で Apache モジュールを実装した場合と遜色ない性能が得られる．

さらに，Apache と並ぶ代表的な Web サーバソフトウェアである nginx [12] にも，提案するアーキテクチャを ngx_mruby として実装した．これにより，Web サービス開発の技術者は，複数の Web サーバソフトウェアの違いを意識することなく，Web サーバの機能拡張を実装できる．

本論文の構成について述べる．2 章では Web サーバソフトウェアの機能拡張について述べる．3 章では 2012 年 4 月にリリースされた組み込みスクリプト言語 mruby と組み込みソフトウェア開発における課題について述べ，4 章

[1] 京都大学情報学研究科
Graduate School of Informatics, Kyoto University, Kyoto 606–8501, Japan
[2] 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606–8501, Japan
[a)] matsumoto_r@net.ist.i.kyoto-u.ac.jp

では mod_mruby のアーキテクチャと適用例について説明する．5 章で mruby スクリプトを Apache の内部機能として実行した場合のパフォーマンス評価を行い，6 章でむすびとする．

2. Web サーバソフトウェアの機能拡張

これまで，Web サーバの機能拡張は，高速かつ軽量に動作することを重視して，C 言語のよる実装が主流であったが，生産性や保守性を考慮して，スクリプト言語で機能拡張を行う手法も提供されている．しかし，従来手法は高速性・省メモリ・安全性の面で課題が残る．そこで，代表的な Web サーバソフトウェアである Apache を例に，従来手法の特徴，および，問題点について言及する．

2.1 Apache モジュールによる機能拡張

Web サーバソフトウェアである Apache は，世界で最もシェアの高いソフトウェア（2013 年 7 月時点）[13] である．Apache の特徴は，最低限の Web サーバ機能をコアとして持ち，その他の機能はコアを改修せず追加できるように，モジュール型の設計 [2] をとっている．Apache の影響により，その他の Web サーバソフトウェアのほとんどが，モジュール型の機能拡張方式をとっている．**図 1** に，Apache のコアとモジュールの概要図を示す．Apache のコアと Apache モジュールの連携は，Apache 独自の API を介して実現されている．モジュールの実装においては，コアに近い実装から Web アプリケーションに近い実装まで，様々な領域の実装が可能となっている．一般的な Web サーバソフトウェアと同様，Apache モジュールによる機能拡張は高速性と省メモリを考慮して C 言語で実装する仕様になっており，Apache 内部で直接 Apache モジュール内の関数をフックする．Perl や PHP，Python，Ruby 等に代表されるようなスクリプト言語で実装する Web アプリケーションは，Apache モジュールでも実装できる．高速に動作することを優先した場合は，Apache モジュールでアプリケーションを開発することも可能である．しかし，一方で，Apache モジュールは C 言語で実装するためにコンパイルが必要である．また，Apache に組み込む必要があり，Apache サーバプロセスの再起動が必要となったり，Web アプリケーションを開発する場合には C 言語程の緻密さを要求しない場面が多く，生産性や保守性の面でスクリプト言語に劣る．Web サービスの大規模・複雑化にともない，高品質・短納期・高い保守性が求められている中で，Web アプリケーションの開発者はスクリプト言語による開発が主流になっているため，C 言語による拡張機能の実装は困難である．

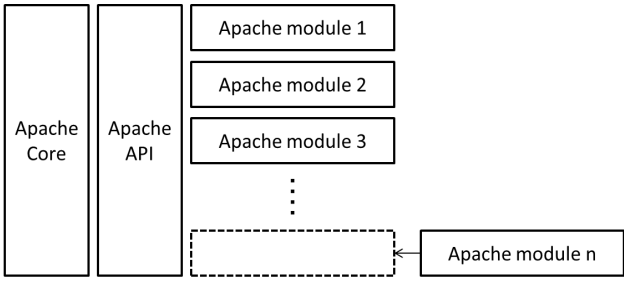

図 1 Apache モジュールの仕組み
Fig. 1 The architecture of Apache modules.

2.2 スクリプト言語による機能拡張

これまで，Web サーバソフトウェアの機能拡張を支援するために，Apache において，スクリプト言語の Perl や Ruby で，Apache モジュール相当の実装を可能とする mod_perl [6] や mod_ruby [7] が開発されてきた．これらの従来手法は，高速に処理するために，インタプリタを複数のスクリプトで共有するが，アプリケーションとしてまったく別々のスクリプト間でもグローバル変数名が干渉し合い，安全に実装できないという問題があった．また，そもそも，Perl や Ruby のインタプリタやライブラリが巨大であるため，mod_php [14] のように Web アプリケーションの実装を主目的として，ライブラリが充実しているほど良い場合には適しているが，高速かつ省メモリに Web サーバそのものの内部処理を拡張することを主目的とした場合には適していない．

C 言語で実装された Apache の内部処理として Ruby や Perl 等のリッチなスクリプト言語を使うには課題がある一方で，軽量組み込みスクリプト言語である Lua [9] が人気を高めてきた．Lua はリオデジャネイロ・カトリカ大学の情報工学科コンピュータグラフィックステクノロジーグループ TeCGraf らによって設計開発された組み込みスクリプト言語である．Lua は C/C++ で実装されたホストプログラムに組み込み，ホストプログラムの一部の処理を Lua スクリプトで実装することを目的に設計されており，高速な動作と高い移植性，組み込みの容易さが特徴である．移植性を高めるため，いったんバイトコードにコンパイルされ，Lua VM 上で実行する方式をとっている．変数に型のないスクリプト言語では最速の言語処理系 [15] だとされている．

Lua を Apache でも利用できるような Apache モジュールの開発が行われてきている．2012 年 2 月に数年ぶりの Apache のメジャーバージョンである 2.4 がリリースされた．Apache2.4 は，シェアを伸ばしてきている nginx に対抗するために，各種パフォーマンス改善や新機能の実装が行われた．新機能の中には，Lua を Apache でも利用できるように，Apache に Lua を組み込むためのモジュール mod_lua [8] が試験的に導入されている．mod_lua によって，Lua スクリプトを Apache の Web コンテンツとして扱うことができ，また，Apache の内部処理として，Lua スクリプトで定義した関数を呼び出すことが可能になっ

ている．Apache の内部処理として Lua スクリプトを実装すれば，スクリプトとしての保守性や開発効率を生かしたまま，これまでの Perl や Ruby よりも高速かつ省メモリに Apache の内部処理を実装することが可能となる．しかし，mod_lua にはいくつか課題がある．第 1 に，Lua スクリプト実行時に，インタプリタに関する情報を保存しておく状態遷移保存領域をスクリプト単位で生成・解放しており，全体として処理速度が遅くなる．状態遷移保存領域は，Lua VM が動作するうえで必要な情報を格納する用途で使われ，生成・解放のコストが非常に高い．Lua の仕様では，1 つの状態遷移保存領域で，複数のスクリプトを共有することが可能であるが，共有してしまうと，状態遷移保存領域の解放ができないため，リクエストごとにサーバプロセスのメモリが増加したり，他のスクリプトのグローバル変数や関数を参照できたりしてしまう．このような実装をとってしまうと，Apache の内部処理を複数のスクリプトで実装した場合，スクリプト上のグローバル変数や関数が干渉し合い，安全に実装できない．そのため，mod_lua では状態遷移保存領域を共有しない実装をとっていると考えられる．一方，mod_perl や mod_ruby では，状態遷移保存領域を事前に初期化しておき，複数のスクリプトで状態遷移保存領域を共有していたが，同様に別のスクリプトのグローバル変数を参照できてしまい，バグのリスクが高い．また，メモリ使用量も多くなってしまう．

以上より，高速性，省メモリ，安全性，および，C/C++ アプリケーションへの組み込みを目的とした軽量組み込みスクリプト言語に着目して，Web サーバの機能拡張をスクリプト言語で行うためには，状態遷移保存領域の確保と解放のコストを減らすことでスクリプトを高速に処理させる必要がある．さらに，Web サーバプロセスのメモリ使用量を低減し，各スクリプトが干渉しないように影響範囲を分離するアーキテクチャが必要である．以降では，それらを実現するためのアーキテクチャ設計について言及する．

3. 組み込みスクリプト言語 mruby

Web サーバソフトウェアの機能拡張において，生産性・保守性を意識しながら高品質かつ短納期に実装する，という課題は，組み込みソフトウェア開発現場における課題と類似している点がある．そこで，組み込みソフトウェア開発現場の課題を解決するために開発されている組み込みスクリプト言語 mruby [11] がある．mruby は組み込み用の処理系であり，通常の Ruby 処理系より省メモリに動作する．また，機能的には Ruby のサブセットであり，ISO/IEC30170 規格に沿って実装されている．

3.1 mruby の概要

mruby は，まつもとゆきひろ氏を中心に開発され，2012 年 4 月 20 日にソースコードが公開された組み込みスクリプト言語である．組み込みスクリプト言語 mruby を C/C++ で実装されたホストアプリケーションに組み込むと，そのホストアプリケーションの一部を Ruby で実装することが可能となる．C 言語での実装が前提となるような組み込み機器上でも，mruby をホストアプリケーションに組み込むことで，Ruby による記述が可能となる．C/C++ アプリケーションが主とすると，mruby は従の関係にあり，C/C++ アプリケーションの専用のパーツとしての使い方が想定されている．スクリプト言語と相性の良いテキスト処理等の実装は Ruby で実装し，高速性や省メモリ等，緻密さが優先される箇所は C 言語で実装する，といった使い方ができる．これによって，大規模，複雑化した組み込みソフトウェア開発を C 言語のみで実装する場合，高品質，短納期，高い保守性を実現するのは困難であったが，mruby は今後その課題を解決するための 1 つの手法になっていくと考えられる．

3.2 mruby と Lua

2 章で言及した組み込みスクリプト言語 Lua は，ネットワークルータの機能拡張や，ゲームに組み込んでキャラクターの AI 等の実装に採用され，最近では iPhone アプリの開発にも利用されてきている．Lua も mruby と同様，C/C++ で実装されたホストアプリケーションに組み込むための API が整備されており，プログラムサイズも比較的小さくなる特徴がある．しかし，Ruby と比べてデフォルトで使用できるライブラリが非常に少ない．そこで，mruby ではそのような問題を解決するために，これまでに多くの技術者によって改善がなされてきた Ruby の言語仕様やライブラリ，Ruby のオブジェクト指向の記述方法を考慮して，Lua よりも比較的ライブラリを充実させ，ある程度のマシンスペックを要求するような組み込みソフトウェアを想定して実装されている．具体的には，mruby が動作した機器として，32 bit の ARM core を搭載し，クロック周波数 84 MHz，SRAM96 KB，プログラム格納用フラッシュメモリ 512 KB の Arduino Due 上での動作実績がある．

3.3 mruby の設計方針

組み込みソフトウェアのための Ruby である mruby は，以下の特徴を持つ．

(1) 組み込み API
(2) 省メモリ
(3) モジュールやクラスが取り外し可能
(4) JIS/ISO を尊重
(5) 移植性
(6) ソフトリアルタイム

(1) の組み込み API に関して，プログラミング言語 Ruby における C API は，1 つのプロセス上で Ruby アプリケーションが単一の仮想マシン上で動作し，そこに機能を追加

するためのものであった．しかし，mrubyでは，組み込みソフトウェアで実装されるような，1つのプロセスに複数の仮想マシンを持ち，それぞれの仮想マシン上でRubyスクリプトを呼び出せる設計になっている．mrubyの組み込みAPIは充実しているため，C/C++で実装されたホストプログラム上でもRubyスクリプト実行までの各種処理過程で，任意のタイミングで構文解析をしたり，バイトコードを生成したり，様々な最適化が容易になる．

(2)，(3)に関して，省メモリを達成するために，mrubyでは，Rubyのソースコードを仮想マシンが解析してバイトコードにコンパイルする機能を，あらかじめ取り外すことができる．

開発中は，コンパイラおよびコード生成機能をリンクしておき，開発完了後は，Rubyのソースコードをあらかじめバイトコードとして生成しておくことで，コンパイラやコード生成機能を不要にすることができる．これにより，100 Kバイト程度プログラムサイズを削減できる．さらに，コアに取り入れられている標準ライブラリから，実行に不必要なモジュールやクラスを取り外すことができる．たとえば，標準出力が不要なプログラムの場合は，標準出力のクラスを取り外したり，配列や連想配列等一部の機能を使わない場合は関連するクラスを取り外したりすることで，さらにメモリを節約することができる．

(4)，(5)に関して，組み込み領域では標準規格を非常に重視するため，mrubyはRubyのISO規格[17]を尊重して設計されている．また，mrubyはC言語で実装されており，移植性を高めるために，1999年に制定されたC言語の国際基準であるC99に準拠した記述にしている．ファイルシステムやさらにはOSがないような環境でも動作するように，mrubyは実装されている．

(6)に関して，組み込みソフトウェアではリアルタイム性が重視される．リアルタイム性とは，処理時間が一定以内に収まることをさしており，mrubyでは，ソフトリアルタイム性を想定して実装されている．従来のRuby処理系は人間に検知できる程度の長時間ガベージコレクションで停止することがあり，リアルタイム用途への障害となっていたが，mrubyはインクリメンタルGCの採用でリアルタイム性を向上させている．

以上のように，mrubyが開発された理由として，組み込みソフトウェアの規模と役割の増大にともなう，開発手法に対する，高品質，短納期，高い保守性が求められていることをあげた．ハードウェアの性能が向上してきている一方で，C言語による開発よりも，生産性や保守性を優先できる分野が増えていくと考える．また，我々のWebサーバ機能拡張に関する研究背景である，Webサービスの大規模・複雑化にともなう，Webサービス開発の高品質・短納期・高い保守性の実現は，mrubyの開発背景と類似していることから，機能拡張のスクリプト言語にmrubyを採用した．

4. mod_mrubyのアーキテクチャと適用例

Webサーバの実装や拡張機能追加は，これまでC言語での開発が主流であり，かつ，Webサーバソフトウェアの内部仕様を詳細に理解している必要があった．また，2章で，スクリプト言語の普及にともない，Webサーバソフトウェアをスクリプト言語で拡張することの重要性を説いた．しかし，これまでの手法は，生産性や保守性を優先した場合においても，リクエストごとに処理されるWebサーバの内部処理にスクリプトを採用するには，高速性，省メモリ，スクリプト間の干渉，および，C/C++アプリケーションへの組み込みを目的とした軽量組み込みスクリプト言語が必要であるという要求を満たせていない．一方で，Webサービスが大規模・複雑化してきているなか，C言語によるWebサーバソフトウェアの拡張は，高品質かつ短納期に開発し，高い保守性を維持しながら運用していかなければならない点において，効率が悪い．

3章で言及したように，現状のWebサーバソフトウェア拡張における問題を組み込みソフトウェア開発における問題と類似しているととらえ，スクリプト言語で機能拡張可能で，高速かつ省メモリで動作し，スクリプト間で干渉しあわないWebサーバの機能拡張支援機構mod_mrubyを提案する．

4.1 mod_mrubyのアーキテクチャ概要

mod_mrubyは，高速性と省メモリの観点からサーバプロセスに軽量に動作する組み込みスクリプト言語mrubyを最適化して組み込み，そのうえで，スクリプトが互いに影響を与えないように分離するアーキテクチャを取った．また，本論文のアーキテクチャはWebアプリケーションだけでなく，Webサーバソフトウェアそのものの内部処理を制御することを主目的としている．Webアプリケーションの実装を主目的とする場合は，言語の軽量さよりも，ライブラリや実装記述方法が充実していることの方が重要だと考えており，そのような用途においては，現行のRuby on Rails[18]やPHP[14]，Java Servlet[19]等の方が適している．mrubyの特徴であるモジュールやクラスの取り外しによって不必要な機能を省略し，mrubyの組み込みAPIを利用することで，生産性や保守性を優先しながらも，省メモリなWebサーバ拡張機構を提供できる．また，これまでは，Apacheのモジュールを C言語で実装後，本番環境と同一の構成を持つコンパイル環境でコンパイルしてからApacheに組み込む必要があり，その場合はApacheの再起動が必要となる．また，コードに変更が必要な場合は，再度コンパイル環境で変更してコンパイル後，再度組み込む必要があるため，保守性が低い．一方，mod_mrubyは，Webサーバ管理者が迅速にリソース管理やアクセス制御等

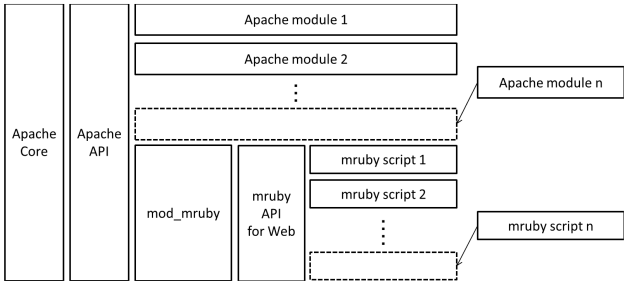

図 2 Apache と mod_mruby の仕組み
Fig. 2 The architecture between mod_mruby and Apache.

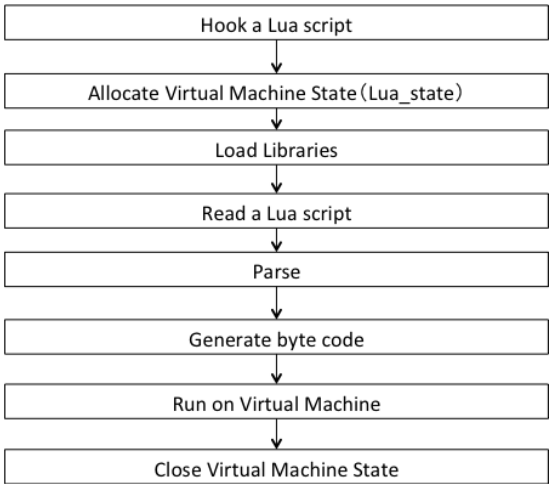

図 3 mod_lua のアーキテクチャ
Fig. 3 The architecutre of mod_lua.

の機能拡張を行えるように，あらかじめ Apache と Ruby スクリプトの処理を連携するインターフェイスを Apache モジュールとして実装しておく．**図 2** に，mod_mruby の仕組みを示す．mod_mruby を Apache に組み込むことによって，Ruby スクリプトに Apache の内部処理を実装できる．mruby と Apache は専用の mruby 用 API ライブラリを介して連携する．mruby 上で記述できないような複雑な処理の場合は，Apache モジュールとして C 言語で実装し，Ruby スクリプトと共存しながら組み込むことも可能である．Ruby スクリプト内の実装の変更も，スクリプトを書き換えることで，書き換え完了以降のリクエストは書き換え後のスクリプトで処理できる．また，スクリプトの変更がそれほど必要とされない状況では，つど Ruby スクリプトを呼び出さずに，サーバ起動時にバイトコードまではコンパイルしておくことも可能である．

以降，高速性，および，省メモリとスクリプト間の干渉の観点から mod_mruby のアーキテクチャの詳細を述べる．

4.2 高速性

2 章で，mod_lua の性能に関する実装の問題を指摘した．**図 3** に Lua スクリプト実行時の mod_lua のアーキテクチャを示す．mod_perl や mod_ruby は，複数のスクリプトで同一の状態遷移保存領域を使用する場合に，別々のアプリケーション間で各スクリプトのグローバル変数を共有してしまう問題があった．しかし，mod_lua ではそれらの問題を解決するために，性能を犠牲にして，スクリプトを実行するごとに状態遷移保存領域を生成し，実行後は状態遷移保存領域の解放するアーキテクチャをとっている．一方，mod_mruby は，状態遷移保存領域を複数のスクリプトで共有することで，高速処理できるアーキテクチャをとった．状態遷移保存領域には，構文解析やインタプリタが持つバーチャルマシン上で動作するバイトコードの生成，その他，バーチャルマシンそのものや変数を保存するハッシュテーブル等，スクリプト実行に必要な情報のほとんどを保存している．そのため，その領域の確保処理は非常に高コストである．**図 4** に mod_mruby のアーキテクチャを示す．リクエスト処理時に，処理コストの高い状態遷移保存領域の再確保やライブラリの読み込み等を実行しないようにするために，Apache のサーバプロセス起動時に，状態遷移保存領域をあらかじめ生成し，ライブラリを読み込んでおく．そして，Apache 起動後，クライアントからリクエストがあると，図 4 のように，リクエストの各種処理フェーズで Ruby スクリプトが Apache からフックされる．そこで，事前に生成されている状態遷移保存領域を呼び出して，その領域上でスクリプトの処理を開始する．リクエスト処理ごとに，Ruby スクリプト自体の構文木解析を行い，バイトコードを生成する．そのバイトコードは，状態遷移保存領域内に存在するバイトコードテーブルに保存される．その結果，スクリプト単位で処理コストの高い状態遷移保存領域の確保やライブラリの読み込みが必要である mod_lua と比較して，高速に処理を行うことができる．しかし，このアーキテクチャは状態遷移保存領域を解放せずに再利用するため，スクリプト処理ごとにメモリが肥大化していく問題と，スクリプト間でグローバル変数を共有し

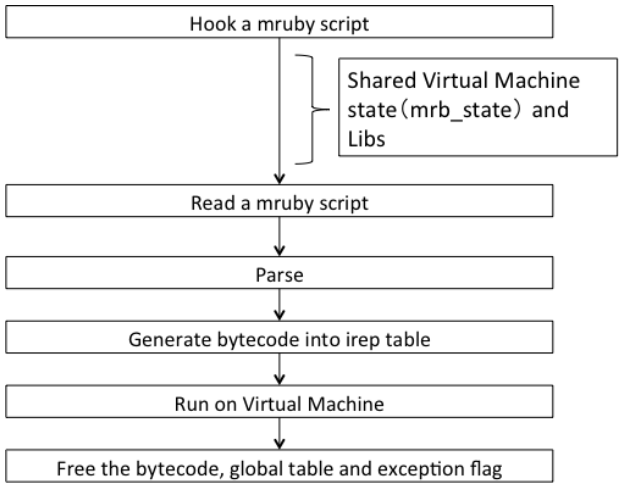

図 4 mod_mruby のアーキテクチャ
Fig. 4 The architecture of mod_mruby.

てしまう問題がある．

4.3 メモリ効率とスクリプト間の干渉の改善

上記の高速性を考慮したアーキテクチャをとった場合の問題を解決する方法を述べる．mruby は，バイトコード生成前後でメモリが大きく増加する．つまり，メモリ消費量が肥大化する主な原因は，コンパイルによって生成されたバイトコードを状態遷移保存領域内のバイトコード保存領域（以降 irep テーブルと呼ぶ）に保存し，次に新たにスクリプトを実行する際には，irep テーブルの保存する領域のインデックス番号をインクリメントして，新しいインデックス番号が指し示す領域にバイトコードを保存していたためであった．mod_mruby では，Apache 起動中であっても Ruby スクリプトを変更することで，Apache の振舞いを即時変更することができるように，基本的にはリクエストごとにコンパイルしてバイトコードを新しく生成する必要がある．つまり，古いバイトコードを irep テーブルに保存しておく必要はないと考えられる．その特徴を考慮して，スクリプト実行時に自動的にバイトコードを解放するようにした．バイトコード生成後，irep テーブルに保存する際にそのバイトコードを保存する領域のインデックス番号を保存しておく．バイトコードを実行した後は，状態遷移保存領域上の irep テーブルから，保存しておいたインデックス番号を元に対象のバイトコードを解放し，インクリメントされた数だけデクリメントするようにした．この処理を mruby の組み込み API に追加することで状態遷移保存領域を再利用しながらも，スクリプト実行前後に増加していたプロセスのメモリ消費量を大幅に低減することができる．

状態遷移保存領域を共有することによるスクリプト間の干渉問題については，状態遷移保存領域内には，例外処理フラグとグローバル変数テーブルが存在する．これらが，スクリプト干渉における主な原因となるため，バイトコードの解放に加えて，例外処理フラグとグローバル変数テーブルから任意のグローバル変数も解放する．これによって，新たな Ruby スクリプトが同一の状態遷移保存領域上で処理されても，古いスクリプトの情報と干渉が起きないようにできる．また，Web サーバの特性上，マルチスレッドによるリクエスト処理等を考慮して，バイトコードやグローバル変数テーブルの保存と解放の間を排他処理することにより，インデックス番号の齟齬が起きないようにした．グローバル変数の解放の手法は，mod_ruby や既存手法からの移行を考慮して，プログラマが手動で記述することにより解放するように実装した．**図 5** に解放のための記述例を示す．平易な記述でグローバル変数を解放することができるため，記述コストは大きく増えない．また，グローバル変数の衝突を完全に防ぐためには，すべてのスクリプト上において，使用したグローバル変数を解放する上記処理を追加する必要がある．その場合の性能面に関して，状態遷移保存領域には，4.2 節で言及したとおり，バーチャルマシンに必要な情報や，その他，シンボルや変数のための多くのハッシュテーブルが存在する．グローバル変数用のハッシュテーブルはそれらの 1 つに過ぎず，さらにはグローバル変数の解放はテーブル内の特定のカラムを解放するのみの処理となるため，状態遷移保存領域全体と比較した場合には非常に小さい処理となり，状態遷移保存領域の共有の利点が失われる程の性能上の問題は生じない．一方で，スクリプト間で変数を受け渡ししたい状況，たとえば，サーバ起動時にデータベースサーバに接続しておいて，その接続情報を別のフェーズで再接続することなく再利用したい場合に，それらの情報を，mod_mruby の機能により安全に受け渡しすることも可能である．

```
$hoge = 1

Apache::global_remove :$hoge
```

図 5 グローバル変数解放の記述例
Fig. 5 Example code to free global variables.

```
LoadModule mruby_module modules/mod_mruby.so
mrubyTranslateNameMiddle /path/to/sample.rb
```

図 6 mod_mruby の Apache 設定例
Fig. 6 Example Apache configuration for mod_mruby.

```
r = Apache::Request.new

r.filename = "/var/www/html/redirect.html"

Apache::return Apache::OK
```

図 7 mruby スクリプト例
Fig. 7 Example code of mruby.

4.4 機能面

mod_mruby によって Apache 内部の処理を Ruby スクリプトで実装するために，Ruby 上から Apache 内部の関数や構造体を操作できるクラスを設計した．これによって，Ruby スクリプト上から様々な Web サーバソフトウェア拡張が可能となる．**図 6**，**図 7** に，Ruby による機能拡張実装例を示す．図 6 は，Apache がリクエストを受けた際に，アクセスのあった URI とファイルを紐づけるフェーズで，Ruby スクリプトをフックするための Apache の設定例である．図 7 は，アクセスのあった URI と特定のファイルを紐づけて，レスポンスにそのファイルの内容を返す処理を Ruby で実装した例である．

Apache と並ぶ代表的な Web サーバソフトウェアである nginx [12] にも，提案するアーキテクチャを ngx_mruby と

して実装した．また，他の Web サーバソフトウェアにも同様の機能を実装予定である．これにより，Web サービス開発の技術者は，Apache や nginx 等，複数の Web サーバソフトウェアの違いを意識することなく，Web サーバの機能拡張を実装できる．また，mruby は Ruby と同様のオブジェクト指向による実装が可能なため，現在 Web サービス開発で普及しているプログラミング言語 Ruby を使って Web アプリケーションを実装したり，ミドルウェアを実装したりしている技術者にとって，Web サーバの機能拡張を実装し易くなると考えている．

5. パフォーマンス評価

mod_mruby が実用に耐えうるかを評価するために，Apache の内部処理を実装した Ruby スクリプトを，mod_mruby を介して実行した場合の性能を評価した．

5.1 サーバプロセスメモリの増加量に関する評価

mod_mruby は，高速性を得るために，状態遷移保存領域を共有しながらも，メモリが増加しないようにバイトコードのみを解放するようにしたアーキテクチャをとった．そのアーキテクチャによるメモリ増加量の低減を確認するため，Apache に mod_mruby を組み込み，図 6，図 7 の処理を行うようにした．そして，バイトコードを解放しない場合と，解放する場合において，それぞれ 5 万リクエスト処理するまでの，サーバプロセスのメモリ増加量を複数回測定し，その平均値を算出した．**図 8** にメモリ増加のグラフを示す．図 8 より，バイトコードの解放をしない場合は，リクエスト処理ごとに Apache の内部で Ruby スクリプトが実行され，そのバイトコードが状態遷移保存領域に保存されてしまい，サーバプロセスの消費メモリが単調増加していることが分かる．それを防ぐために，バイトコードのみを解放することで，サーバプロセスのメモリが増加していないことが分かる．これにより，バイトコードのみを解放するアーキテクチャで，十分にメモリ増加量を低減できていることが分かった．また，図 8 より Apache サーバプロセスのメモリ使用量 5,000 kB であるが，mod_mruby を組み込む前の Apache のサーバプロセスのメモリ利用量が約 4,400 kB であり，mod_mruby 組み込みに要するメモリ使用量は約 600 kB である．現在の Web サーバの搭載メモリが数 GB 単位であることを考慮すると 600 kB は非常に少ないといえる．

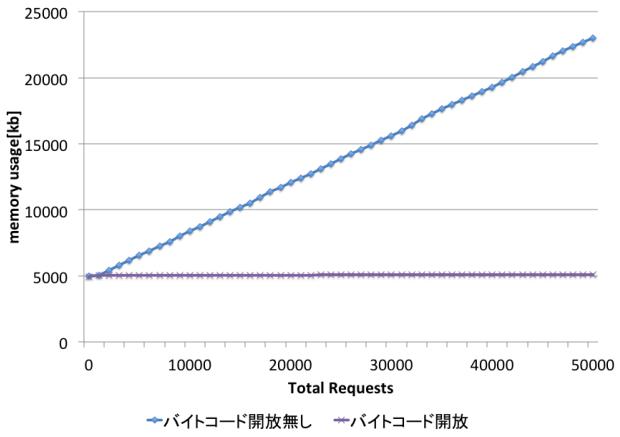

図 8 バイトコード解放によるメモリ増加量
Fig. 8 Memory increment when freeing bytecode.

5.2 高速性に関する評価

次に，既存の機能拡張手法と高速性の比較を行った．状態遷移保存領域が作成され，コンパイルされることによる処理の影響を最大化するため，実行するスクリプトはクライアントがどのような URI にアクセスしても，すべてのアクセスに対して hello world の文字列を返すようなコストの低い処理を実装とした．評価においては，同様の処理を従来のように C 言語によって実装した Apache モジュール mod_hello と，mod_perl による Perl スクリプトでの機能拡張実装，mod_ruby による Ruby スクリプトでの実装，mod_lua による Lua スクリプトでの実装と比較を行った．**表 1** は，クライアントとサーバマシンの性能を示している．クライアントマシンから，サーバマシンに対して，ab コマンドにより，同時接続数 100，総接続数 10 万のパラメータでリクエストを送信し，サーバマシンが 1 秒間に処理できたレスポンス数を複数回計測しその平均値を算出した．同時接続数のパラメータは，サーバの Apache や OS のチューニングがボトルネックにならないように，予備実験により適切だと思われるパラメータを設定した．

表 2 は 1 秒間に処理できたレスポンス数 (responses/sec) の結果を有効数字 6 桁により示している．パフォーマンス評価の結果，mod_mruby が mod_lua よりも性能が非常に高いことが分かった．これは，状態遷移保存領域を，mod_lua はスクリプト実行ごとに生成しているのに対し，mod_mruby は複数のスクリプトで共有しているためだと

表 1 テスト環境
Table 1 Hardware configuration of test environments.

クライアント	
CPU	Intel Core2Duo E8400 3.00GHz
Memory	4GB
NIC	Realtek RTL8111/8168B 1Gbps
OS	CentOS 5.6
サーバ	
CPU	Intel Xeon X5355 2.66GHz
Memory	8GB
NIC	Broadcom BCM5708 1Gbps
OS	CentOS 5.6
Middleware	Apache/2.2.3

表 2 パフォーマンス評価と各手法の特徴
Table 2 Performance comparison and characteristics of each method.

	言語	Reponses/sec
mod_hello	C	9861.17
mod_perl	Perl	3346.38
mod_ruby	CRuby	4769.04
mod_lua	Lua	5209.11
mod_mruby	mruby	9021.54

考えられる．また，Ruby スクリプトに Apache の内部処理を実装し，mod_mruby を介して実行しても，従来手法である mod_hello が 9,861.17 response/sec であるのに対して，mod_mruby は 9,021.54 response/sec であり，mod_lua の 5,209.11 response/sec と比較して，非常に高速に動作していることが分かった．また，既存の mod_perl や mod_ruby は mod_mruby と同様事前に状態遷移保存領域を初期化していながら，mod_mruby よりも性能が出ていない．これは，mod_ruby で使われている CRuby インタプリタは非常に高機能で規模が大きく（数 MByte のバイナリサイズ），mruby インタプリタ（数百 Kbyte）と比較して多くのクラスやメソッドを使用可能であり，構文解析やバイトコードの生成時（以降コンパイルとする）のクラスやメソッドの検索のコストが非常に大きくなるためである．C 言語で書かれた Web サーバの実装の一部をスクリプト記述で代替するために，CRuby のような高機能で規模の大きいインタプリタを組み込むのは，リクエストごとにスクリプトのコンパイルが必要となることを考えると非常にコストが高い．そのため，mod_ruby は状態遷移保存領域を共有しているが，コンパイル処理がボトルネックととなり，大幅に性能が低下する．さらに，mod_ruby が mod_lua よりも性能が出ていない理由は，CRuby 程度のインタプリタの規模になった場合，状態遷移保存領域の確保・解放よりも，コンパイル時の処理がボトルネックとなるためである．一方で，mod_lua のインタプリタの規模は，mod_mruby と同等かやや小さいものとなり，インタプリタの規模が同等の状況においては状態遷移保存領域を共有するかどうかが性能に大きく影響を与える．そのため，mod_mruby が mod_lua よりも高速に動作する．さらには，予備実験において，mod_mruby の実装を一時的にリクエストごとに状態遷移保存領域の確保を行うようにして同様の実験を行った場合，約 2,000 response/sec 程度の値しか出なかったことからも状態遷移保存領域の確保と解放のオーバヘッドが大きいことが分かる．また，今回の実験においては状態遷移保存領域を共有するかどうかが大きなボトルネックとなっており，予備実験においても，事前にライブラリロードすることによる低減はほとんど確認できなかった．しかし，ライブラリの事前ロードは，今後 mruby に組み込んでおくライブラリが増加していった場合に，事前にロードしておくことで，つどロードするよりも性能劣化が低減されることが期待される．

以上より，文字列を出力するのみの軽量な処理においては，mod_mruby のサーバプロセスへのインタプリタ組み込みの最適化のアーキテクチャによって，インタプリタの処理部分は，動的コンテンツを扱ううえでほとんどボトルネックにならないと想定される．また，サーバプロセスのメモリ増加量も気にする必要がない．スクリプトの保守性や開発のしやすさを考慮した場合，サーバ拡張のための 1 つの選択肢になりうると考えられる．

6. まとめ

本論文では，Web サービスの大規模・複雑化を考慮して，Web サーバの機能拡張を支援するために，スクリプト言語で機能拡張が可能でありながら，高速かつ省メモリで動作する Web サーバの機能拡張支援機構 mod_mruby を提案した．また，高速性を優先した場合に生じる，スクリプト間でグローバル変数が干渉し合う問題を同時に解決した．mod_mruby によって，Ruby スクリプト上に Apache の内部処理を記述することで，即時処理を拡張できる．その結果，Web サービス開発に関わる技術者が Web アプリケーション等を Ruby で開発する延長で，容易に Web サーバ自体を拡張できるようになると考えている．また，nginx を mruby によって機能拡張できる ngx_mruby の存在により，Apache や nginx の Web サーバソフトウェアの違いを Ruby スクリプトで吸収しながら機能拡張が可能になる．

今後の課題として，ngx_mruby だけでなく，代表的な Web サーバやキャッシュサーバソフトウェアを mod_mruby とできるだけ同じ記述で機能拡張できるインターフェイスを実装予定である．その結果，複数の Web サーバソフトウェア上に，mruby による機能拡張インターフェイスを組み込むことで，Web サーバソフトウェア間の実装の違いを mruby で吸収し，Web サーバの機能拡張を Ruby での実装に統一できると考えている．

謝辞 mod_mruby を実装するにあたり，有益なコメントをいただいたまつもとゆきひろ氏，増井雄一郎氏，松本泰弘氏に感謝する．

参考文献

[1] Prodan, R. and Pstemann, S.: A survey and taxonomy of infrastructure as a service and web hosting cloud providers, *2009 10th IEEE/ACM International Conference Grid Computing*, pp.17–25 (Oct. 2009).
[2] Apache モジュール一覧，入手先 ⟨https://httpd.apache.org/docs/2.2/ja/mod/⟩.
[3] 原 大輔，尾崎亮太，兵頭和樹，中山泰一：Harache：ファイル所有者の権限で動作する WWW サーバ，情報処理学会論文誌，Vol.46, No.12, pp.3127–3137 (2005).
[4] 松本亮介，川原将司，松岡輝夫：大規模共有型 Web バー

チャルホスティング基盤のセキュリティと運用技術の改善，情報処理学会論文誌，Vol.54, No.3, pp.1077–1086 (2013).
[5] 松本亮介，岡部寿男：スレッド単位で権限分離を行う Web サーバのアクセス制御アーキテクチャ，電子情報通信学会論文誌，Vol.J96-B, No.10 (2013).
[6] Alexandros, L. and Roussopoulos, N.: Generating dynamic content at database-backed web servers: Cgi-bin vs. mod_perl, *ACM SIGMOD Record*, Vol.29, No.1, pp.26–31 (2000).
[7] mod_ruby, available from ⟨https://github.com/shugo/mod_ruby⟩.
[8] The Apache Software Foundation: Apache Module mod_lua, available from ⟨http://httpd.apache.org/docs/trunk/mod/mod_lua.html⟩.
[9] Roberto Ierusalimschy, Waldemar Celes and Luiz Henrique de Figueiredo, The Programming Language Lua, available from ⟨http://www.lua.org⟩.
[10] The Apache Software Foundation: Apache HTTP Server Project, available from ⟨http://httpd.apache.org/⟩.
[11] NPO 法人軽量 Ruby フォーラム，入手先 ⟨http://forum.mruby.org/⟩.
[12] nginx: nginx, available from ⟨http://nginx.org/ja/⟩.
[13] Netcraft: July 2013 Web Server Survey, available from ⟨http://news.netcraft.com/archives/2013/07/02/july-2013-web-server-survey.html⟩.
[14] PHP, available from ⟨http://php.net/⟩.
[15] Brent Fulgham: The Computer Language Benchmarks Game, available from ⟨http://benchmarksgame.alioth.debian.org/⟩.
[16] 経済産業省：平成 22 年度「地域イノベーション創出研究開発事業」に係る委託先の公募について，入手先 ⟨http://www.meti.go.jp/policy/local_economy/tiikiinnovation/22fy_inoberd.html⟩.
[17] ISO: ISO/IEC 30170:2012 Information technology – Programming languages – Ruby, available from ⟨http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=060&ics3=&csnumber=59579⟩.
[18] Ruby on Rails, available from ⟨http://rubyonrails.org/⟩.
[19] Java Servlet 3.0 Specification, available from ⟨http://jcp.org/en/jsr/detail?id=315⟩.

松本 亮介 （学生会員）

2008 大阪府立大学工学部情報工学科卒業．同年ファーストサーバ株式会社入社．レンタルサーバ（ホスティング）の基盤技術に関わる研究・開発・運用に従事．2012 年 4 月より，京都大学情報学研究科博士課程在籍．

岡部 寿男 （正会員）

1988 京都大学大学院工学研究科修士課程修了．同年京都大学工学部助手．同大型計算機センター助教授等を経て 2002 より同学術情報メディアセンター教授．博士（工学）．2005 より国立情報学研究所客員教授．インターネットアーキテクチャ，ネットワーク・セキュリティ等に興味を持つ．システム制御情報学会，日本ソフトウェア科学会，IEEE，ACM 各会員．