プログラム変換による動的 Web ページ高速化の一手法*

竹辺 靖昭

コグニティブリサーチラボ 研究開発部
Yasuaki_Takebe@crl.co.jp

概要

我々は，部分評価の手法を応用し，動的 Web ページの生成を高速化するシステム PHP-Mix を開発してきた．しかし，PHP-Mix は 2 段階の部分評価をベースにしているため，様々な間隔で更新されるデータを含むページやアクセス時にはじめて内容が確定するページには適用が難しい．本研究では，こうした問題を解決するため多段階の generating extension を用いた部分評価手法を採用する．この手法では，スクリプト中に埋め込まれたキャッシュの更新間隔および実行時に得られるパラメタを指定するディレクティブの情報を元に束縛時解析を行い generating extension を生成する．現在までに，実験的な処理系を作成し，動的 Web ページの生成を高速化できることを確認している．

1. はじめに

現在の多くの Web サイトでは，Web サーバ上で動作するスクリプトで動的に Web ページを生成する処理が行われている．この方式では，ブラウザからの処理要求があるたびにスクリプトが実行され，データベースへのクエリなどの処理が行われる．このため，アクセスが集中する Web ページを動的に生成するとシステムへの負荷が高くなってしまうという問題がある．

この問題を解決するため，生成された Web ページの一部領域を Web サーバ側でキャッシュするシステムが開発されている．一般に，動的 Web ページにはあまり内容が変わらない領域と頻繁に内容が変わる領域が混在している．このため，内容が変わらない領域を部分的にキャッシュすることにより高速化が期待できる．

ただし，こうしたシステムで部分キャッシュを行うためには，Web ページを生成するためのスクリプトをフラグメントに分割して作成しなければならない．このため，既存の Web ページに適用しようとすると大幅な変更が必要になってしまう．また，分割したスクリプトを実行するためにはそれに対応した処理系が必要である．

これとは別に，我々は，部分評価の手法を応用し，動的 Web ページの生成を高速化するシステム PHP-Mix を開発してきた[1]．PHP-Mix では，ページ中のあまり内容が変わらない領域を部分評価によってあらかじめ生成し高速化を行う．この手法では，スクリプトを分割する必要がなく，実行も一般に普及している処理系で行うことができる．

しかし，PHP-Mix は伝統的な部分評価の手法をベースにしているため，Web ページの生成は常に 2 段階で行われる．動的 Web ページ中のデータは一般には様々な間隔で更新されると考えられるため 2 段階では十分とはいえない．また，スクリプトから得られる情報だけを用いて部分評価を行っているため，パーソナライズ機能を持つページのようにアクセス時にはじめて内容が確定するページに適用するとほとんど部分評価ができないことがある．部分キャッシュを行う多くのシステムでは実行時に得られるパラメタの値ごとにキャッシュを行う方法が用意されており，こうしたページでも効果を得ることができる．

こうした問題を解決するため，本研究では 2 段階の部分評価手法の代わりに多段階の generating extension を用いた部分評価手法を採

* **A Method to Speed Up Dynamic Web Pages Using Program Transformation Techniques**, Yasuaki Takebe, R&D Division, Cognitive Research Laboratories, Inc., Tokyo.

用する．また，実行時にも部分評価を行うことによりパラメタの値ごとのキャッシュと同等の機能を実現する．

このプログラム変換はWebページの開発によく使用される手続き型のスクリプト言語 PHP[5]のサブセットを対象とする．この言語では，部分キャッシュでの指定方法にならい，キャッシュの更新間隔およびパラメタを指定するディレクティブをスクリプト中に埋め込むことができる．本手法では，このディレクティブの情報を元に束縛時解析を行い，generating extensionを生成する．

論文の構成は以下の通りである．第2節では，既存の動的 Web ページの高速化手法および本研究の基本となる部分評価手法について述べる．第3節では，本研究で用いた部分評価手法について説明する．第 4 節では，本研究の部分評価手法をWebサイトに適用するためのシステムの概要について説明する．第5節では，実験的に作成した動的 Web ページに本手法を適用した結果を評価する．最後に，第6節で関連する研究について，第 7 節でまとめと今後の課題について述べる．

2. 従来の技術

この節では本研究の背景として既存の動的 Web ページの高速化技術および部分評価手法について述べる．

2.1. ページフラグメントのキャッシュ

動的 Web ページはあまり内容が変わらない領域を動的に組み合わせることによって作られていることが多い．これらの領域をフラグメントとしてWeb サーバ側でキャッシュすることにより動的Webページを高速化する技術が開発されている．

Challengerらは，Webページをフラグメントに分割し，フラグメントとWebページの依存関係に従ってデータの更新を伝播する手法により，キャッシュのヒット率を向上することに成功している[2]．

Microsoft 社が開発した ASP.NET[3]では部分キャッシュ用のディレクティブが用意されており，フラグメントごとにキャッシュの有効期限等のポリシーを設定することができる．また，クッキー等のブラウザから渡されるデータやブラウザの種類ごとに別のバージョンのキャッシュを作成することができる．このため，パーソナライズ機能をもつページへの適用も容易である．

以下にこのディレクティブの例を示す．

```
<%@ OutputCache Duration="10"
                VaryByParam="foo" %>
```

Duration 属性はキャッシュの有効期限を示す．VaryByParam 属性はHTTPのGETメソッドによって送られるパラメタの値ごとに別バージョンのキャッシュを作成することを示す．この例ではキャッシュの有効期限は10秒であり，fooパラメタの値ごとに別のバージョンのキャッシュが作成される．

2.2. 部分評価

部分評価とは，プログラムへの入力の一部だけを与え，その入力に特化したプログラムを生成するプログラム変換の一手法である．生成されたプログラムは残りの入力を処理するものになる．

部分評価の定式化として，以下の記述がよく用いられる．

$$[\![p]\!]\,[in_1, in_2] = [\![[\![mix]\!]\,[p, in_1]]\!]\,in_2$$

ここで，mix は部分評価器，p はプログラム，in_1，in_2 は入力を表す．$[\![p]\!]$ in はプログラム p に入力 in を与えることを表す．部分評価器に与えられた入力 in_1 を静的，残りの in_2 を動的であるという．

部分評価では，プログラムの実行は静的なものと動的なものとの2段階であるが，これをn段階に拡張したものも考えられている．Glück らは，n 段階の部分評価を効率よく行うものとして，多段階の generating extension を用いる手法を提案している[4]．以下では[4]に従い多段階の generating extension の定式化を行う．

まず，2段階の generating extension の定式化を以下に示す．

$$\text{p-gen} = [\![\text{gegen}]\!]\,p$$
$$[\![\text{p-gen}]\!]\,in_1 = [\![mix]\!]\,[p, in_1]$$

ここで，p-gen はプログラム p に対する generating extension，gegen は generating extension 生成器を表す．部分評価器は，プログラムと静的な入力から特化したプログラムを生成す

```
program ::= stmts
stmt ::= expr ; | echo expr ; | if (expr) stmt else stmt | while (expr) stmt | { stmts }
        | '// cache' cache_attribs '// endcache'
cache_attrib ::= duration = duration | param = id
expr ::= constant | var | var = expr | expr op expr | id ( exprs )
var ::= $ id | $ id indexs
index ::= [ expr ]
```

図 1. 対象言語の構文

るが，generating extension 生成器は，プログラムから，静的な入力から特化したプログラムを生成するプログラムを生成する．特化したプログラムを生成するプログラムを generating extension という．

多段階の generating extension の定式化は以下になる．

$$
\begin{aligned}
&\text{p-mgen} = [\![\text{mgegen}]\!]\ \text{p}\\
&\text{p-mgen}_1 = [\![\text{p-mgen}]\!]\ \text{in}_1\\
&\text{p-mgen}_2 = [\![\text{p-mgen}_1]\!]\ \text{in}_2\\
&\cdots\\
&\text{p}_{n-1} = [\![\text{p-mgen}_{n-2}]\!]\ \text{in}_{n-1}\\
&\text{out} = [\![\text{p}_{n-1}]\!]\ \text{in}_n
\end{aligned}
$$

ここで，out はプログラム p に入力 $\text{in}_1, \ldots, \text{in}_n$ を与えて得られる出力である．多段階の generating extension 生成器 mgegen によって，n 入力のプログラム p の実行を n 段階に分けて行うことができる．

2.3. PHP-Mix

PHP-Mix は動的 Web ページを生成するスクリプトを部分評価することにより高速化を行っている．部分評価の際に，定数だけでなく更新の頻度が比較的低いデータも静的と解析することにより，ページ中のあまり内容が変わらない領域を生成することができる．PHP-Mix は，更新の頻度が低いデータの更新にあわせて部分評価を再度行うことにより，生成されるページとデータとの整合性を保っている．

ただし，PHP-Mix には 2 段階の部分評価手法を用いていることによる問題点がある．

まず，Web ページ中のデータがさまざまな頻度で更新される場合にも 2 段階でページの生成を行わなければならない．このため，更新の頻度が非常に低いデータであっても同じページ内により更新の頻度が高いデータがある場合には高いほうの頻度に合わせて部分評価を行わなければならない．

また，PHP-Mix の部分評価手法はスクリプトから得られる情報だけを用いているため，パーソナライズ機能を持つページのようにアクセス時にはじめて内容が確定するページには適用が難しい場合がある．PHP-Mix では，変数が実行時にとると思われる値をユーザがあらかじめスクリプト中のディレクティブに指定することで，こうしたページもある程度は効果的に部分評価することができる．しかし，変数のとる値が前もってわかっている場合にしかこの手法は適用できない．

最後に，PHP-Mix を Web サイトに導入する際には，部分評価器をサーバに配置しなければいけないという問題がある．これは部分評価器には部分評価したいプログラムと静的な入力を同時に与えなければいけないためである．静的な入力である更新の頻度が低いデータは Web サイトを運用しているサーバでしか得られない．

信頼性が要求されるサーバには実績の少ないソフトウェアの導入は避けられる傾向がある．なるべく既存の処理系のみで運用できる手法のほうが望ましいといえる．

3. 本研究の部分評価手法

この節では，本研究で用いる部分評価手法について述べる．対象言語の仕様，束縛時解析，generating extension の生成の順に説明を行う．

3.1. 対象言語

本研究では，Web ページの開発によく使用される手続き型のスクリプト言語 PHP のサブセットに

キャッシュの更新間隔とパラメタを指定する構文を追加した言語を対象とする．対象言語の構文を図1に示す．

簡単のため関数の定義は考慮していない．PHP ではライブラリ関数があるため，これらの呼び出しは行うことができる．ただし，Web ページの出力を行う手段を echo 文のみに限定するため，Web ページの出力を行うライブラリ関数は使えないものとする．また，制御構文には goto 文はない．この点は PHP も同じである．

この言語は，PHP と同様に弱い型つけの言語であり，以下のデータ型が扱える．

- スカラー値(数値，文字列)
- 配列
- リソース値

このうち，スカラー値と配列はスクリプト中にリテラルとして表現することができるが，リソース値は表現できない．PHP では，ある種のライブラリ関数の返却値がリソース値になる．例えば，ファイルハンドルやデータベースからのクエリの結果などがリソース値である．

キャッシュの更新間隔およびパラメタの指定は cache 文によって行う．duration 属性にはキャッシュの更新間隔，param 属性にはパラメタとなる変数の名前を指定する．param 属性は省略が可能である．

以下に，指定されたユーザ ID からデータベースに格納されているユーザ情報を表示するスクリプトの例をあげる．

```
$uid = $_GET["uid"];

// cache duration=120 param=uid
$result = pg_exec(
    "SELECT * FROM users WHERE uid=$uid");
$user = pg_fetch_array($result, 0);
echo $user["name"];
echo $user["address"];
// endcache
```

cache 文により，ユーザ ID をパラメタとして更新間隔120秒でのキャッシュを行う指定がされている．$_GET は PHP の特殊な変数であり，HTTP の GET メソッドによって送られるパラメタの値が格納されている．

3.2. 束縛時解析

3.2.1. 束縛時

束縛時とは，スクリプト中の要素がいつ評価されるかを示す値である．スクリプト中の式および文がとる束縛時を求める解析を束縛時解析という．本手法の場合は，束縛時は0以上の整数になる．値が大きいほうがより動的になる．

束縛時解析の結果を表すのに，式や文にアノテーションをつけた多段階のプログラムがよく用いられる．以降の説明でも，式や文に下線を引き，添え字で束縛時を示したものを多段階のスクリプトとし，束縛時解析の結果を表すのに用いる．

3.2.2. 束縛時解析のための前処理

まず，前処理として以下の処理を行う．

- キャッシュの更新間隔に対応する束縛時の計算
- 変数のリフト可能性の解析
- 生存解析

キャッシュの更新間隔に対応する束縛時は，スクリプト中の各 cache 文に対して，duration 属性が大きいものから順に0以上の整数を割り当てることによって求められる．duration 属性の値が同じ cache 文が複数ある場合には同じ束縛時を対応させる．例えば，duration 属性の値がそれぞれ120，30，30 であった場合，対応する束縛時はそれぞれ 0，1，1 となる．

また，duration 属性の値に対応する束縛時のうちで最大のものに1を加えた値を *bt_max* とする．上記の例の場合，*bt_max* は 2 である．

変数のリフト可能性の解析は，スクリプト中の各変数がリソース値を取る可能性があるかどうかの解析である．部分評価では，静的な変数がリテラルで表現可能な値をとっていれば，それを変換結果中に埋め込むことができる．この操作をリフトという．変数がリソース値を取る可能性がなければリフト可能，リソース値を取る可能性があればリフト不能と解析する．この解析は，スクリプト中で呼び出されているライブラリ関数の戻り値の型を元にし

$$\tau \vdash const : 0 \qquad \tau \vdash v : \tau(v)$$

$$\frac{\tau \vdash e : b \quad liftable(e)}{\tau \vdash lift_b(e) : b+1} \qquad \frac{\tau \vdash v : b \quad \tau \vdash e_i : b}{\tau \vdash \underline{v[e_1]\ldots[e_n]}_b : b}$$

$$\frac{\tau \vdash e_1 : b \quad \tau \vdash e_2 : b}{\tau \vdash \underline{e_1\ op\ e_2}_b : b} \qquad \frac{\tau \vdash e_i : b \quad duration\_bt \le b}{\tau \vdash \underline{f(e_1,\ldots,e_n)}_b : b}$$

$$\frac{\tau \vdash e_1 : b \quad \tau \vdash e_2 : b \quad control\_bt \le b}{\tau \vdash \underline{e_1 = e_2}_b : b}$$

$$\frac{\tau \vdash e : b}{\tau \vdash \underline{e;}_b : b} \qquad \frac{\tau \vdash e : bt\_max}{\tau \vdash \underline{echo\ e;}_{bt\_max} : bt\_max}$$

$$\frac{\tau \vdash e : b}{\tau \vdash \underline{if\ (e)\ stmt_1\ else\ stmt_2}_b : b} \qquad \frac{\tau \vdash e : b}{\tau \vdash \underline{while\ (e)\ stmt}_b : b}$$

図2. 束縛時解析のための型規則

て行うことができる.

変数がリフト可能かどうかの解析結果を用いることにより, 式がリフト可能かどうかを求めることができる.

3.2.3. 式および cache 文以外の文の解析

まず, cache 文以外の文および式に対する束縛時解析について述べる.

束縛時解析には, 抽象解釈を用いるものと型規則を用いるものがある[6]. 型規則を用いるものは, スクリプト中の式がとる束縛時を型と考え, この型に対する制約を型規則によって与える. この制約を解くことにより束縛時を求めることができる. ただし, 本研究で対象としている言語には副作用があるため, 通常の型規則による制約に加えて, 制御構造を考慮した制約も満たす必要がある.

束縛時解析の解が満たすべき型規則を図2にあげる. 本研究で実装した処理系では, アドホックな手法でこれらの制約を満たすように各変数の束縛時を求めている.

τは束縛時の環境であり, $\tau(v)$は変数 v の束縛時を表す.

b は束縛時をとる変数である.

liftable は, 前処理で行ったリフト可能性の解析結果を元に, 式がリフト可能かどうかの判定を行う関数である. この制約により, リフト可能な式のみにリフト操作が認められる.

duration_bt は, 関数呼び出しを含む最も内側の cache 文の duration 属性に対応する束縛時である. ライブラリ関数の呼び出しは, 常に *duration_bt* 以上に動的な束縛時を持つ. cache 文の外側では *duration_bt* は *bt_max* とする.

control_bt は, 代入文を含むすべての if 文および while 文の束縛時のうちで最大のものである. 制御文中で代入が行われた変数は, 常にその制御文の束縛時よりも動的な束縛時を持つ.

例えば, 以下のスクリプトにおいて, `$y` に対しては定数のみが代入されているが, echo 文で使用される`$y` の値は`$x` に依存している.

```
if ($x == 0) $y = 1; else $y = 2;
echo $y;
```

このため, `$y` の束縛時は`$x` よりも動的に解析する必要がある. *control_bt* による制約は, こうした解析を行うために必要になる.

echo 文は Web ページの出力を行う文であるため, ブラウザからアクセスを受けた段階で実行する必要がある. このため, 束縛時は常に *bt_max* になる.

3.2.4. cache 文の解析

duration 属性や param 属性に応じて部分評価を行うために, 本手法では cache 文ごとに束縛時環境を与えている. 変数は cache 文の外部と内部

で異なった束縛時を持つことができる.

本手法では, cache 文を含む文の解析を2つのフェーズに分けて行う. まず, cache 文の外部の束縛時環境を決定するための解析を行う. このとき, 正しく generating extension を生成できるように, cache 文の外部と内部の環境が以下の制約を満たすようにする. ここで, τ_1, τ_2 はそれぞれある cache 文の外部の環境, 内部の環境を表すとする.

1. すべての変数 v について $\tau_2(v) \leq \tau_1(v)$.
2. cache 文の先頭で生存している変数および cache 文の末尾で生存しているリフト不能な変数 v について $\tau_2(v) = \tau_1(v)$.

cache 文の先頭で生存している変数については, cache 文の外部でその変数がとる値が cache 文の内部でも用いられるため, 同じ段階で評価する必要がある. したがって外部と内部で異なる束縛時を持つことはできない. cache 文の末尾で生存している変数については, リフト可能な変数である場合, generating extension の生成の際に明示的な代入文を cache 文の末尾に挿入することにより, 外部と内部で異なった束縛時を持っていても対応することができる. リフト不能な変数の場合にはこうした対応ができないため同じ束縛時を持つ必要がある. 制約2はこれを表している.

このはじめのフェーズの解析により, cache 文の外部の環境が決まるとともに内部の環境も仮に求まる.

次のフェーズでは, cache 文の内部が, param 属性に指定されている変数に関して実行時に部分評価可能かどうかを判定し, 可能ならばそれに応じた cache 文の内部の束縛時環境を解析結果として採用する.

この解析は以下の手順で行う. τ_1, τ_2'はそれぞれ cache 文の外部の環境, 内部の環境を表すとする. また, param 属性に指定されている変数を v, duration 属性に指定されている更新間隔に対応する束縛時を *duration_bt* とする.

1. param 属性に変数が指定されていない場合, または $\tau_1(v) <$ *duration_bt* が成り立つ場合には, cache 文の内部は実行時に部分評価ができないと判定し, はじめに求めた τ_2 を解析結果として採用する.
2. v 以外の変数で, cache 文の先頭で生存している変数および cache 文の末尾で生存しているリフト不能な変数 w について, τ_2'は以下の条件を満たすものとする.

   - $\tau_1(w) \geq \tau_1(v)$ ならば $\tau_2'(w) = \tau_1(w) + 1$
   - $\tau_1(w) < \tau_1(v)$ ならば $\tau_2'(w) = \tau_1(w)$

   v については以下を満たすものとする.

   - $\tau_2'(v) \geq \tau_1(v)$

3. 2の条件のもとで cache 文の内部の束縛時解析を行い, τ_2'を求める. このとき, *bt_max* は cache 文の外部の解析で用いたものよりも1大きいとする.
4. 3 の解析の結果, $\tau_2'(v) = \tau_1(v)$ ならば, cache 文の内部は実行時に部分評価が可能と判定し, 求めた τ_2'を解析結果として採用する. $\tau_2'(v) > \tau_1(v)$ の場合は, はじめに求めた τ_2 を解析結果として採用する.

条件2で $\tau_2'(w)$を外部の環境よりも1段階動的にしているのは, 後に生成する generating extension において cache 文の内部は実行時に部分評価されるため, 実行が1段階多く進むためである. $\tau_1(w) < \tau_1(v)$となる変数 w に関しては, 実行時の部分評価よりも前の段階で評価が行われるため, 1段階動的にする必要はない.

3.2.5. 実行時の部分評価結果の生成と無効化

本手法では, cache 文は 2 種類の束縛時を持つ. 2種類の束縛時は, それぞれ実行時に行った部分評価結果を生成するタイミングと無効にするタイミングに相当するものである. これをそれぞれ生成時, 無効化時と呼ぶことにする.

ある cache 文が持つ生成時と無効化時は以下のように求めることができる. ここで, cache 文の外部の環境をτ, param 属性に指定されている変数を v, duration 属性に指定されている更新間隔に対応する束縛時を *duration_bt* とする.

- cache 文の生成時は$\tau(v)$である.
- 無効化時 *it* は, 以下の条件を満たす最小の整数とする.

```
include "genlib.php";

gen_start(1);

$tmp0 .= gen_expr_stmt(
  gen_assign_expr(
    gen_var_expr("a", 1),
    gen_indexed_var_expr("_GET", array(gen_lift("a", 1)), 1), 1), 1);
$b = "b";
$tmp0 .= gen_echo_stmt(gen_var_expr("a", 1), 1);
$tmp0 .= gen_echo_stmt(gen_lift($b, 1), 1);

gen_end(1);
```

図 3. generating extension の例

a. $it \geq duration\_bt$ である.
b. cache 文の先頭で生存している変数および cache 文の末尾で生存しているリフト不能な変数 w で, $it < \tau(w) < \tau(v)$ となるものは存在しない.

cache 文の生成時が $\tau(v)$となるのは, cache 文の内部の部分評価が行われるのと v が評価されるタイミングと同じであることによる.

無効化時に関して条件 b を設けている理由を, 例をあげて説明する. 以下のスクリプトでは, $a の束縛時は *bt_max* とする.

```
// cache duration=10
...
// endcache
// cache duration=1000 param=a
echo $a;
// endcache
```

このスクリプトでは, 2 つめの cache 文内部の処理は$a にのみ依存している. そのため, 1 つめの cache 文の実行結果にかかわらず, $a の評価時に得られる部分評価結果は同じである. このため, 2 つめの cache 文の部分評価結果は, 指定された duration 属性にしたがって 1000 秒ごとに無効化すればよい.

この場合, 条件 b での変数 w に相当するものはないため, 無効化時は *duration_bt* となる.

これに対して, 以下のスクリプトでは, 2 つめの cache 文の部分評価結果が 1 つめの cache 文の実行結果に依存している. このため, 1 つめの cache 文の実行結果が変わる 10 秒ごとに 2 つめの cache 文の部分評価結果も無効化する必要がある.

```
// cache duration=10
$b = time();
// endcache
// cache duration=1000 param=a
echo $a;
echo $b;
// endcache
```

この場合は, $b が変数 w に相当するため, 無効化時はτ($b)と求めることができる.

3.3. Generating Extension の生成

3.3.1. Generating Extension の構造

本手法では, PHP のコードを生成する関数を PHP で作成し, 束縛時解析の結果をもとに, これらのコード生成関数を呼び出す PHP コードを生成することにより, generating extension を生成している.

例として, 以下のスクリプトに対応する generating extension を図 3 に示す.

```
$a = $_GET["a"];
$b = "b";

// cache duration=60
echo $a;
```

```
gen_var_expr(name, 1) = "$name"
gen_var_expr(name, n) = "gen_var_expr(name, m)"   (n > 1, m = n - 1)
```

図 4. 変数を生成するコード生成関数

```
gen_while_stmt(cond, cond_name, stmt, stmt_name, 1) =
    "while (cond) {
        $stmt_name = ¥"¥";
        stmt
        $stack_top .= $stmt_name;
     }"

gen_while_stmt(cond, cond_name, stmt, stmt_name, n) =
    "$cond_name = cond;
     gen_stmt_start(stmt_name);
     stmt
     gen_stmt_end();
     $stack_top .= gen_while_stmt(cond, cond_name, stmt, stmt_name, m)"   (n > 1, m = n - 1)
```

図 5. while 文を生成するコード生成関数

```
echo $b;
// endcache
```

gen_で始まる関数がコード生成関数である. 生成結果は$tmp0 に格納され, gen_end 関数の呼び出し時に出力される. ". ="は PHP の文字列追加演算子である.

コード生成関数の最後の引数は 0 以上の整数であり, 生成されるコードが何段階目に実行されるのかを示している. 0 段階目に実行されるコードは generating extension 中に埋め込まれている.

3.3.2. 式を生成する Generating Extension

式を生成するコード生成関数は, [4]の多段階のコード生成関数と同様のものである. ただし, 本手法では PHP で実装しているため, S 式ではなく PHP の式を表す文字列を出力する.

コード生成関数は各構文に対して定義している. 例として変数を生成するコード生成関数を図 4 に示す.

いずれの構文に対する関数も, 最後の引数が 2 以上の時は, 最後の引数を 1 減らした関数呼び出しを表す文字列を返却し, 最後の引数が 1 になったときにコードを生成する. また, gen_lift 関数は, 引数として与えられた値を表すリテラルを表す文字列を出力する.

多段階の式に対応するコード生成関数の呼び出しを生成することにより, 式を生成する generating extension を生成することができる.

3.3.3. 文を生成する Generating Extension

PHP には文があるため, 関数の呼び出しのみでは部分文を持つ文を生成することはできない. 本手法では, 部分文を持つ文の生成結果を一時変数に格納するコードと, それらの生成結果から文全体を生成するコード生成関数の呼び出しを生成することにより, 文を生成する generating extension を生成している.

例えば, while 文を生成する generating extension は以下の形になる.

```
$tmp1 = 条件式を生成する式;
gen_start_stmt("tmp2");
$tmp2 .= while 文の中の文 1 を生成する式;
...
$tmp2 .= while 文の中の文 m を生成する式;
gen_end_stmt();
$tmp0 .= gen_while_stmt(
```

gen_cache_stmt(*stmt*, *stmt_name*, *timestamp*, *param*, 0, *it*) =
"$*name* = gen_cache_name(*timestamp*, $*param*);
if (!gen_cache_exist($*name*)) {
stmt
gen_write_cache($*name*, $*stmt_name*);
}
include ¥"$*name*¥";"

gen_cache_stmt(*stmt*, *stmt_name*, *timestamp*, *param*, *gt*, *it*) =
"gen_start_stmt(*stmt_name*);
stmt
gen_end_stmt();
$*stack_top* .= gen_cache_stmt(*stmt*, *stmt_name*, *timestamp'*, *param*, *gt'*, *it'*);"

(*timestamp'* = 現在時刻 (*it* = 1 の時)
= *timestamp* (*it* ≠ 1 の時)
gt > 1, *gt'* = *gt* - 1, *it'* = *it* - 1)

図 6. cache 文に対応するコード生成関数

```
$tmp1, "tmp1", $tmp2, "tmp2", n)
```

gen_stmt_start および gen_stmt_end は部分文の範囲を指定するための関数である. gen_stmt_start により, グローバルなスタックに引数の文字列がプッシュされる. 文を生成するコード関数では, このスタックのトップの文字列を名前とする変数に結果を代入する文を生成する. gen_stmt_end はグローバルなスタックから名前をポップする.

例えば,

```
gen_start_stmt("tmp2");
$tmp2 .= gen_echo_stmt(
  gen_var_expr("i", 2), 2);
gen_end_stmt();
```

を実行した後の$tmp2 の値は,

```
$tmp2 .= gen_echo_stmt(
  gen_var_expr("i", 1), 1);
```

という文字列になる. この文字列中の$tmp2 を生成するためにスタックのトップの値を用いている.

文を生成するコード生成関数は, 最後の引数が 2 以上の時には上の形を持つ文の列を表す文字列を返却する. 例として while 文を生成するコード生成関数を図 5 に示す.

文に関しても, 多段階のスクリプトに対応するコード生成関数の呼び出しを生成することにより, generating extension を生成することができる.

3.3.4. cache 文に対応する Generating Extension

束縛時解析によって実行時の部分評価が可能と判定された場合, cache 文からは実行時に部分評価を行う文を生成する generating extension を生成する. そうでない場合は cache 文の内部の文に対応する generating extension を通常の文の場合と同様に生成する.

実行時の部分評価を行う文は以下の形になる.

```
$name = gen_cache_name(timestamp, $param);
if (!gen_cache_exist($name)) {
  $tmp1 .= cache 文の中の文 1 を生成する式;
  ...
  $tmp1 .= cache 文の中の文 m を生成する式;
  $tmp1 .= 明示的な代入文を生成する式;
  gen_write_cache($name, $tmp1);
}
include "$name";
```

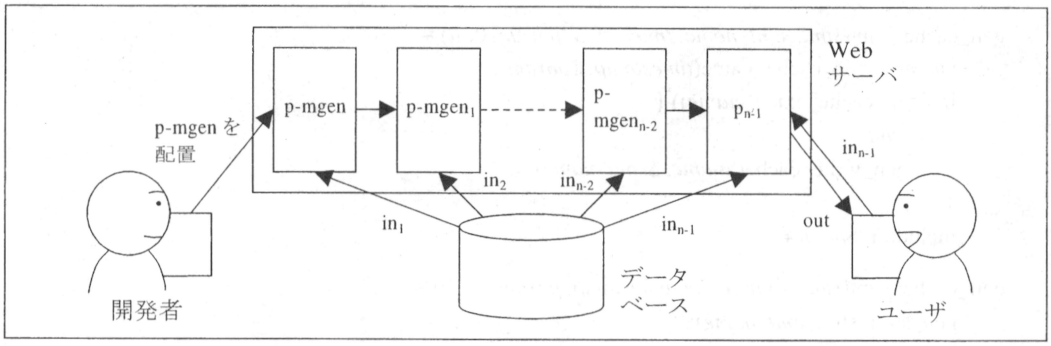

図 7. Web サイトへの適用方法

timestamp は実行時の部分評価結果の有効性を示すためのタイムスタンプ，$param は cache 文の param 属性に指定された変数である．gen_cache_name は，これらの値から部分評価結果が格納されるファイルの名前を決定する．

まだ対応する部分評価結果が存在しない場合には，cache 文の中の文を生成する generating extension を実行し，結果をファイルに書き込む．ファイルは PHP の include 文によってスクリプト中に動的に読み込まれ，実行される．

cache 文は他の文と異なり，生成時と無効化時という 2 つの値をとる．このうち無効化時を上記の *timestamp* を生成するために利用することによって，実行時の部分評価により生成されたコードの無効化のタイミングを調節する．

図 6 に cache 文に対応するコード生成関数を示す．他のコード生成関数とは異なり，最後の 2 つの引数はそれぞれ生成時，無効化時に対応した値になる．

it が 1 の時のみ *timestamp* に現在時間を設定し，その他の段階では同じ値を使う．*timestamp* が変わらない限り gen_cache_name は同じ値を返すため，一度生成したコードが使い続けられる．

cache 文に関しても，多段階のスクリプトに対応するコード生成関数の呼び出しを生成することにより，generating extension を生成する．ただし，cache 文の場合，同じ変数の束縛時が内部と外部で異なる場合があるため，束縛時をそろえるために明示的な代入文を cache 文内部の文の末尾に挿入する．例えば，param 属性に指定されている変数を v, cache 文の外部および内部の環境をそれぞれ τ, σ としたとき，$\tau(w) \geq \tau(v)$ となる変数 w

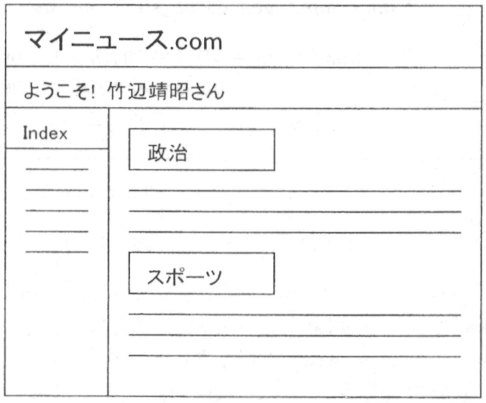

図 8. 測定に用いた Web ページ

への明示的な代入文は以下の多段階の式で表すことができる．

$$\underline{w}_{\tau(w)+1} = \underbrace{lift_{\tau(w)}(\cdots(lift_{\sigma(w)}(\underline{w}_{\sigma(w)}))\cdots)}_{\tau(w)+1}$$

ここで，左辺の w の束縛時を 1 段階動的にしているのは，実行時の部分評価により実行が 1 段階多く進むためである．

4. Web サイトの構成

前節までで述べたプログラム変換の手法の Web サイトへの適用方法を図 7 に示す．duration 属性により指定された間隔で，対応する段階以降の generating extension を順に実行する．Web サイトの開発者が，ソースのスクリプトを generating extension に変換してからサーバに配置することにより，サーバ側に必要なのは PHP の処理系だけ

にすることができる.

現時点では, generating extension を定期的に実行するためのシステムは特に用意していない.

Generating extension 生成器の実装は Java で, コード生成関数の実装は PHP で行い, それぞれの規模は 3050 行, 350 行になった.

5. 評価

5.1. ニュースサイトへの適用

本研究の手法により, 動的な Web ページを生成する負荷がどの程度低減されるかを測定するため, [1]で測定に用いたニュース一覧表示ページへの適用を行った. ただし, 本研究で対象としている PHP のサブセットで記述できるように処理内容に変更を加えている.

ページのデザインを図 8 に示す. このページは, 認証したユーザが興味を持つカテゴリのニュースの一覧をデータベースより取得し表示する機能を持っている.

まず, このページを生成するスクリプトが本手法によりどのように変換されるかを見てみる. ページ生成スクリプトの概要は以下のとおりである.

```
// cache duration=3600 param=ユーザ
if (認証が行われていたら) {
  興味あるカテゴリのリストを取得;
} else {
  デフォルトのカテゴリのリストを取得;
}
// endcache
for (カテゴリ in カテゴリのリスト) {
  // cache duration=60 param=カテゴリ
  そのカテゴリのニュースの一覧を表示;
  // endcache
}
```

cache 文により, 興味あるカテゴリのリストの取得処理とニュース一覧の表示処理の更新間隔を設定している. 興味あるカテゴリは, 一度設定したらあまり変更しないと考えられるため, ニュースよりも更新間隔を長く設定している.

このスクリプトから generating extension を生成し, 2 段階評価すると, 各 cache 文の処理を実行時に

	(1) カテゴリ部分評価なし	(2) カテゴリ部分評価あり	(3) ニュース一覧のみ	(4) cache 文なし
所要時間(秒)	10.48	6.65	8.59	13.74

表 1. 測定結果

部分評価するスクリプトを生成することができる. カテゴリのリストの取得処理をユーザに関して部分評価すると, 結果は, カテゴリのリストを格納する変数への配列定数の代入文だけになる. また, ニュース一覧の表示処理をカテゴリに関して部分評価すると, 結果はニュース一覧を内容とする文字列定数の表示処理だけになる. これにより, 実行時の部分評価結果がすでに生成されている場合には, データベースへのアクセスを行わずにページを生成することができるようになった.

次に, このページに対し, (1)カテゴリのリスト取得処理がまだ部分評価されていない場合と, (2)カテゴリのリスト取得処理の部分評価結果がすでにある場合とで性能評価を行った. 比較のために, (3)ニュース一覧の表示処理のみに cache 文の指定を行った場合と, (4)cache 文の指定を全く行わなかった場合の性能も測定した.

性能測定には, Pentium3/1.13GHz の PC 互換機上で Apache 1.3.26, PostgreSQL 7.1.3, PHP 4.2.3 および phpA 1.3.3r1 を使用した. Apache に付属の ApacheBench プログラムをクライアントから実行し, ページを 1000 回取得するのにかかった時間を測定した. なお, 通常の ApacheBench プログラムには一定の URL をアクセスする機能しかないが, この測定では 1 回のアクセスごとに異なったユーザの ID を用いるように変更を加えている.

測定の結果を表 1 に示す. cache 文の指定を行った場合には, いずれの場合にも(4)よりは性能が向上することが確認できた.

(1)では, カテゴリの取得を行った後に, 部分評価結果をファイルに書き込み, include 文により実行する必要がある. このため(3)よりもかなり性能が低下している. 本手法では生成した部分評価結果が文字列になるため, 初回の実行では構文解析等の PHP 実行系内部の処理が必要になる. これが性能低下の一因ではないかと考えている.

実際のアクセス状況では同じユーザが繰り返し

アクセスを行うことは少ないため，(2)のようになることはない．ただし，ユーザがカテゴリのリストを変更する頻度は非常に低いと予想されるため，定期的に無効化を行わずにすめば部分評価結果が再利用される割合を高くすることは可能だと考えられる．そのためには，[1]が備えているデータの更新に応じて部分評価を行う機能を用意することが必要になるだろう．

5.2. 問題点

本研究で実装した処理系は実験的なものであるため，実行時の部分評価結果の読み込みおよび書き込みの処理に排他制御を行っていないなど，必要な処理が抜けている部分がある．今後は，こうした処理も実装した上で性能を測定していく必要がある．

Generating extension および実行時の部分評価を用いていることによる問題としては，部分評価時のエラー処理がある．

例えば，実行時の部分評価の際にデータベースアクセスのエラーが生じた場合，その結果が部分評価結果に反映され再利用されてしまう．

また，本手法では，静的に無限ループするスクリプトの部分評価は停止しない．[1]では実行ステップ数に上限を設けた部分評価器を用いることにより無限ループに対処している．本手法では通常の PHP で generating extension を実行しているため，こうした対応をとるのは難しい．

6. まとめ

動的Webページの生成を高速化するために従来我々が用いていた 2 段階の部分評価に代えて多段階の generating extension による部分評価を用いる手法を提案した．この手法では，スクリプト中に埋め込まれたキャッシュの更新間隔とパラメタの指定を利用して束縛時解析を行い generating extension を生成する．

本手法により，様々な間隔で更新される情報を含む Web ページやアクセス時にはじめて内容が確定するページにもプログラム変換による高速化手法を適用することが期待できる．実験では，2 種類の頻度で更新される情報を含みアクセス時にはじめて内容が確定するページで一定の性能向上が得られることが確認できた．

また，本手法は generating extension を用いているため一般に普及している PHP の処理系のみで実行が可能であり，従来手法と比べて導入が容易である．

現段階では本手法はPHPのごく一部の言語仕様しか考慮していない．これはなるべくフルセットの PHP に近づけるように拡張していく必要がある．また，本研究で実装した処理系はまだ実験段階のものであるため，今後は現実の応用を考慮に入れ実装方法を検討する必要がある．

参考文献

[1] 竹辺 靖昭, 湯淺 太一: 部分評価を応用した動的 Web ページのキャッシュ機構, 情報処理学会論文誌: プログラミング, Vol.43 No.SIG8 (PRO15), pp98-109 (2002)

[2] J. Challenger, P. Dantzig and A. Iyengar: A Scalable System for Consistently Caching Dynamic Web Data, *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies* (1999)

[3] ASP.NET, `http://www.asp.net/`

[4] R. Glück, J. Jørgensen: Efficient Multi-level Generating Extensions for Program Specialization, *Lecture Notes in Computer Science*, 982, pp259-278 (1995)

[5] PHP, `http://www.php.net/`

[6] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall (1993)