もぶろげっと: 画像情報を含む blog 記事検索システム

Mobloget: A Retrieval System for Texts and Images in Blogs

井原 伸介　　林 貴宏　　尾内 理紀夫*

Summary. We have developped Mobloget: a retrieval system for texts and images in blogs, and have opened the system to public through medias. Mobloget consists of a retrieval module and a search module. The retrieval module collects blogs from the web using a crawler which is implemented by customizing an open source crawler called JSpider, extracts texts and images from the blogs, and stores them to a database. After a user inputs keywords as a search query to Mobloget, the search module displays a thumbnail image with a text-summary for each relevant blog. We have investigated users' opinions regarding Mobloget and have confirmed that thumbnail images help users' understanding to content of blogs.

1　はじめに

blog(Weblog) と呼ばれる Web ページが急速に普及している[1]．blog とは頻繁に更新され，更新された順に記事が並べられていく形式の Web サイトのことである．記事の投稿に技術的な知識を必要とせず，加えて多くの企業により無料でサービスが提供されているため，個人の WWW での情報発信をますます盛んにさせている．

blog の普及に合わせて，blog を対象とした検索エンジンが多数公開されるようになった．たとえば technorati[2]や blogPulse[1]，bulkfeeds[3]などを挙げることができる．また，ニュースサイトの記事と blog 記事とを合わせて収集し，双方を関連づけてユーザに提示するシステム [4] や，話題の注目度や評価表現の多寡を検索条件に取り入れた blog 検索システム [2] など，blog のデータを利用した様々なシステムが提案されている [5].

blog は，その出現からしばらくは，ネットサーファー達がインターネット上の様々な Web ページを記録していくのに使われることが主だった．実際，Weblog という名称は “Web” と “Log” を足し合わせて作られたものである．しかし，急速な普及に伴ってその使われ方は変容し，現在では日記のような個人的な文章を書くのに使われる傾向が強い．

そのような状況の中で，画像情報を主体とした blog が数多く見られるようになっている．携帯電話などのモバイル機器を使ってテキストや写真を投稿する「moblog(モブログ)」や，デジタルカメラで撮影した写真にコメントや解説を添えて投稿する「photolog(フォトログ)」が出現し，次第にその数を増やしている．

blog には主に個人によって記事が投稿される．そのため，それらの記事に含まれる画像には，個人の生活や興味関心に密接したものが多く，「blog に投稿される画像を検索したい」という要求があると考えられる．また，一般的な blog 検索エンジンでは，検索結果として blog 記事のタイトルや要約等を表示するが，もしそこに画像を併せて表示することができれば，ユーザにとって，その blog 記事を読みに行くかどうか判断するための情報が増え，blog 記事の内容をより直感的に理解できるようになると期待できる．

そこで我々は，blog に投稿される画像情報に注目し，画像情報を含む blog 記事を収集し，それらを検索するシステムを構築した．また，本システムを「もぶろげっと[4]」として，メディアを通じて一般に公開した[5].

* Shinsuke Ihara, Takahiro Hayashi and Rikio Onai, 電気通信大学

[1] ブログ・SNS の現状分析及び将来予測, 総務省, http://www.soumu.go.jp/s-news/2005/pdf/050517_3_1.pdf

[2] technorati, http://technorati.com/

[3] bulkfeeds, http://bulkfeeds.net/

[4] もぶろげっと, http://mobloget.jp/

[5] 本システム公開のプレスリリースは以下のメディアで取り上げられた.
internet.com, http://japan.internet.com/webtech/20050127/4.html
IT Media, http://www.itmedia.co.jp/enterprise/articles/0501/27/news072.html
InternetWatch, http://internet.watch.impress.co.jp/cda/news/2005/01/27/6231.html
デジカメ Watch, http://dc.watch.impress.co.jp/cda/other/2005/01/27/835.html
ASCII24, http://ascii24.com/news/i/soft/article/2005/01/28/653922-000.html
週刊アスキー, 2005 年 3 月 1 日号, p114

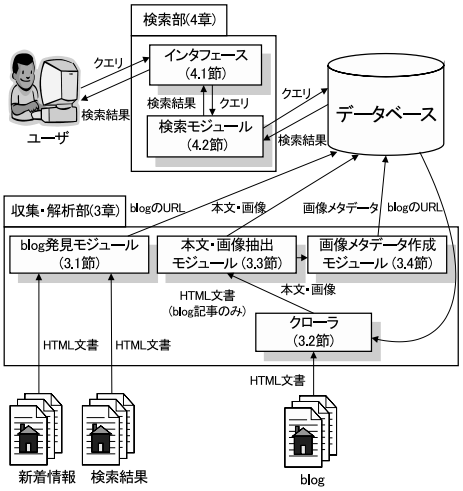


図 1. システム概要

本論文では，本システムの設計と実装について述べる．また，意見調査の結果から本システムを評価する．以下，第 2 章では本システムの概要を述べる．第 3 章では blog 記事の収集機能と解析機能について，第 4 章ではインタフェースと検索機能について述べる．第 5 章では本システムの評価を行う．第 6 章で本論文をまとめる．

2 システムの概要

本システムの概要を図 1 に示す．本システムは，大きく分けて収集・解析部 (3 章) と検索部 (4 章) から構成される．

収集・解析部は，Web から blog 記事を収集し，本文・画像の抽出などの解析を行う．本システムで「blog 記事」とは，1 投稿分の記事のみを含む Web ページを指す．blog のトップページや，月単位やカテゴリ単位で作成される Web ページは，多くの場合複数記事を含むため,「blog 記事」に含めない．収集・解析部は以下の 4 つのモジュールで構成される．

1. **blog 発見モジュール**(3.1 節)—クローリングの対象となる blog の URL を Web から収集し，データベースに登録する．本システムで「blog の URL」とは，その blog のトップページと，その blog の提供する RSS の URL とを指す．
2. **クローラ**(3.2 節)—データベースに登録された blog の URL に対してクローリングを行い，新しく投稿された blog 記事を収集する．
3. **本文・画像抽出モジュール**(3.3 節)—クローラによって収集された blog 記事を受け取り，その本文と，本文に含まれる画像とを抽出する．また，本文のメタデータとして，blog 記事のタイトルと投稿された時刻の抽出を行う．
4. **画像メタデータ作成モジュール**(3.4 節)—本文・画像抽出モジュールの抽出した画像についてメタデータを作成する．

検索部では，収集・解析部によって抽出された blog 記事の本文に対する全文検索の機能を，本システムのユーザに提供する．検索部は以下の 2 つのモジュールで構成される．

1. **インタフェース**(4.1 節)—ユーザからクエリを受け取り，検索結果を HTML 文書として提示する．
2. **検索モジュール**(4.2 節)—インタフェースからクエリを受け取り，データベースに蓄積された blog 記事の本文に対して全文検索を行いその結果を返す．

現在対応している blog サービスおよび blog ツールの一覧は，本システムの Web サイト[6]にて公開している．2005 年 8 月現在，本システムは約 35 万の blog の URL と約 240 万の blog 記事をデータベースに保持している．1 日に新たに収集・処理される blog 記事の件数は約 4 万である．システムは完全に自動化されており，人手を必要とすることなく動作している．

3 収集・解析部

3.1 blog 発見モジュール

blog 発見モジュールは，各 blog サービスによって提供されている，直近に更新された blog を一覧表示している Web ページ等を定期的に巡回し，HTML 文書から blog のトップページの URL を抽出する．また，得られた URL のリストをデータベースと照合し，未登録のものがあれば，その情報をデータベースに追加する．

次に，発見された blog のトップページの各々について，RSS の URL の有無を調べ，RSS の URL が存在する場合は，合わせてデータベースに登録する．すべての blog に RSS の URL が存在するとは限らないので，RSS の URL を見つけられないこともある．そこで，以下の 1. から 3. の順に処理を行い，3. まで実行しても RSS の URL を見つけられない場合は，RSS が存在しないものとし，データベースには RSS の URL は空白として登録する．

1. blog のトップページを取得し，link タグによって RSS が指定されているか調べる
2. 1. で RSS が指定されていない場合，トップページの URL から blog サービスを特定することを試みる．多くの blog サービスでは，出力する RSS のファイル名を固定名としており，

[6] http://mobloget.jp/?mode=document&name=faq

表 1. 典型的な RSS のファイル名

atom atom.xml index.rdf index1_0.rdf
index.xml index.php index.rss
rss rss.php rss.xml

blog サービスを特定できれば，RSS の URL を決定することが可能である．

3. 2. で RSS の URL を決定できない場合，典型的な RSS のファイル名 (表 1) を blog のトップページの存在するディレクトリに付け足し，取得できるかどうかを調べる．

3.2 クローラ

3.2.1 クローラのスケジューリング

クローラは，データベース内の blog の URL のリストを参照し，blog 記事のクローリングを行う．これらの blog は，その更新間隔に大きなばらつきがある．毎日のように新しい記事が投稿される blog がある一方で，全く更新のない blog もある．よって，すべての blog に対して，一定の時間間隔でクローリングを行うのは効率的でない．本システムでは，クローリングする時間間隔を blog ごとに設定している．また，この値はクローリングを行うたびに調整される．

blog 発見モジュールが新しい blog の URL をデータベースに登録するとき，時間間隔として初期値 (24 時間) が設定される．以後，クローラは設定された時間間隔に基づいてクローリングを行い，前回のクローリング以後に新しい記事が投稿されていたかどうかによって時間間隔を調整する．新しい記事が見つかった場合，時間間隔は 0.9 倍され，見つからなかった場合，1.1 倍される．また，時間間隔は 1 時間より短くはしないこととしている．

3.2.2 blog 記事のクローリング

blog 記事のクローリングには 2 つの方法がある．HTML 文書を収集する方法と，RSS を用いる方法である．HTML 文書を収集する方法では，トップページからリンクを辿って，そのディレクトリ以下に存在する HTML 文書をすべて収集し，その中から blog 記事だけを取り出す．blog に存在するすべての blog 記事をもれなく収集することが可能であるが，リンクを辿るために HTML 文書をパースする必要があり，処理コストが大きい．一方，RSS を用いる方法では，まず RSS を取得し，そこから取り出した blog 記事の URL のリストから収集を行う．RSS は Web サイトの見出しや要約を記述する XML ベースのフォーマットであり，blog 記事の URL を容易に取り出すことが可能であるため，処理コストが小さい．しかし，RSS には直近に更新された一定件数の blog 記事の URL しか含まれないため，その blog に存在するすべての blog 記事を収集できないという欠点がある．また，RSS を出力しない blog に対してこの方法を用いることはできない．

そこでクローラは，初回のクローリング時には HTML 文書に対する収集を行い，blog に存在するすべての記事データを取得する．次回以降のクローリング時には，まず RSS を取得してデータベースと照合し，新しい記事が投稿されていればその HTML ファイルを取得する．RSS を提供していない blog に対しては，毎回 HTML 文書に対する収集を行う．

3.2.3 blog 記事の判定

クローリング以降の処理は blog 記事単位で行われる．HTML 文書に対する収集を行った場合，収集されたデータには blog 記事以外の HTML 文書を含むため，そこから blog 記事のみを取り出す必要がある．ここでは blog 記事に典型的に見られる URL にマッチするような正規表現のセットを作成しておき，その中の一つでも URL にマッチすればそれは blog 記事であると判定する．クローラは blog 記事であると判断されたもののみをデータベースに追加し，それ以外の HTML 文書は破棄する．

3.2.4 クローラの実装

本システムのクローラは，JSpider[7] 0.5.0 をベースにして実装した．JSpider はオープンソースで開発されている中規模な Web クローラである．純粋な JAVA で実装されており，柔軟なクラス構造を持つため用途に応じたカスタマイズに適している．RSS のパースには，オープンソースで開発されている XML パーサ Informa[8] を用いた．

我々が開発したクローラのソースコードは，2005 年 8 月現在，脚注の URL[9] より入手可能である．

3.3 本文・画像抽出モジュール

3.3.1 本文の抽出

本文・画像抽出モジュールは，クローラが収集した blog 記事を受け取り，まず blog 記事の本文を抽出する．本システムで「本文」とは，blog 記事に含まれるテキストのなかで，blog の筆者によって書かれた部分であるとする．収集された状態の blog 記事には，多くの場合，ナビゲーション用のリンクや広告など，本文以外のテキストが含まれている．本システムでは，本文以外のテキストは検索対象として適切ではないと考える．そこで，本モジュールによって抽出された本文のみをデータベースに登録する．

[7] JSpider, http://j-spider.sourceforge.net/
[8] Informa, http://informa.sourceforge.net/
[9] http://www.seman.cs.uec.ac.jp/MoblogetCrawler0.5.zip

一般に blog 記事は，テンプレートに本文を埋め込むことによって作成される．そのため，同一のテンプレートから作成された blog 記事であれば，本文以外の部分は共通であることが期待できる．そこで，本モジュールの処理は blog 単位で行うものとし，blog 記事同士を互いに比較し，共通部分を取り除くという考え方で本文抽出を行う．また，ある blog がたった一つしか記事を持たない場合は，blog 記事同士の比較ができないため，その blog 記事に対しては以後 3.3.2，3.3.3 の各節で述べる処理は行わない．blog 記事に対する本文抽出は，以下の手順で行う．

1. blog 記事の HTML から，script タグとコメントを除去する
2. blog 記事の中には，自身について記述された RDF[10]が HTML 内に埋め込まれているものがある．RDF が存在すれば正規表現を用いてそれを抽出し，3.3.3 で述べる本文メタデータの作成のために，別に保存しておく
3. blog 記事を無作為に 2 つ選び，組にして UNIX のコマンド diff を実行し，それぞれについて得られた出力を本文とする．

3.3.2 画像の抽出

続いて，3.3.1 で抽出された本文から，HTML タグの情報をもとに画像を抽出する．本モジュールでは，図 2 に示す 3 つのパターンでリンクされた画像を，その blog 記事に含まれるものと見なす．(i) は img タグを用いて本文中に直接画像を埋め込むパターンであり，もっとも典型的に見られるものである．(ii) は a タグを用いて本文中のテキストに画像へのリンクを貼るパターンである．(iii) は img タグで埋め込まれた画像に a タグを用いて画像へのリンクを貼るパターンである．(iii) のパターンに関しては，img タグで埋め込まれた画像をリンク先画像のサムネイル画像（縮小画像）であると見なす．よって，リンク先画像のみを本文に含まれるものとして扱う．また，絵文字やレイアウト調整のための小さな画像を除去するため，縦横の大きさがどちらか一方でも 100 ピクセルを下回るものは除外している．また，検索部で使用するため，抽出された画像のサムネイル画像を作成しておく．

本システムは blog 記事とその画像の対を収集するものであるため，本文中に画像が一つも含まれていない記事は，データベースへ登録しない．

3.3.3 本文メタデータの抽出

3.3.2 に続いて blog 記事のタイトルと投稿された時刻の抽出を行う．以下のような処理を行う．

[10] Resource Description Framework (RDF), W3C Semantic Web Activity, http://www.w3.org/RDF/

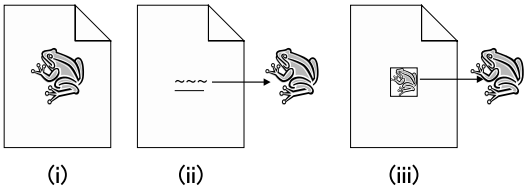


図 2. 画像リンクのパターン

表 2. 画像メタデータの項目

項目名	内容
url	画像の URL
type	画像のデータ形式
width	画像の横の大きさ
height	画像の縦の大きさ
pre	画像の前方のテキスト
post	画像の後方のテキスト
alt	画像の alt 属性

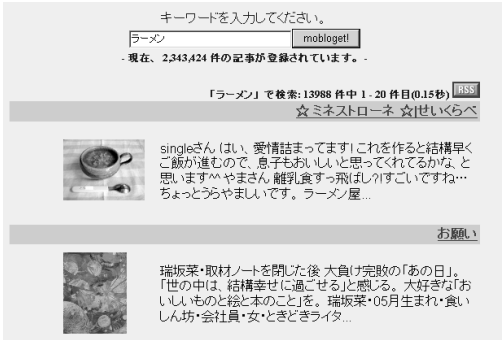
1. その blog 記事について RDF が抽出されていれば，rdf:Description タグの dc:title 属性，dc:date 属性をそれぞれタイトル，時刻とする．
2. RDF が抽出されていなければ，正規表現によってタイトルと時刻の抽出を試みる．タイトルの抽出は blog サービス，blog ツールごとに処理を切り分ける実装をしている．日付の抽出はすべての blog 記事に対して同一の正規表現を用いている．日付の抽出に用いている正規表現を付録 A に示す．
3. 1. と 2. でタイトルと時刻を抽出できなかった場合，タイトルには空白文字列を，日付には UNIX 紀元 (1970/01/01 00:00:00) を設定する．

3.4 画像メタデータ作成モジュール

画像メタデータ作成モジュールは，本文・画像抽出モジュールから受け取った画像について表 2 のような情報を抽出し，メタデータを作成する．メタデータは独自に定義した XML ベースのフォーマットである．

pre と post については，本文中のタグをすべて除去した後，その画像の前後にあるテキストを取り出している．テキストの長さはそれぞれ最大 64 文字である．alt については，その画像の img タグに alt 属性が設定されていれば，そのテキストをそのまま用いる．ただし，(iii) のパターンでは，img タグで埋め込まれた画像の alt 属性を用いる．

現状では，画像メタデータは作成するのみで利用していない．現在，画像メタデータを取り入れた検索手法を検討しているところである．



(a) 通常モード

(b) 画像だけモード

図 3. 検索結果のスクリーンショット

画像メタデータの例を付録 B に示す.

4 検索部

4.1 インタフェース

インタフェースは Web サーバ上で動作する Web アプリケーションとして実装されている．ユーザは Web ブラウザを用いて本システムの Web サーバにアクセスし，入力フォームにクエリを入力することによって検索を行う.

検索結果のスクリーンショットを図 3 に示す．本システムでは，検索結果を「通常モード」(図 3(a)) と「画像だけモード」(図 3(b)) の 2 通りで表示することができる．通常モードでは検索結果に含まれる個々の記事について，そのタイトル，要約，サムネイル画像を表示する．要約の作成には Namazu[11] の機能を使っている．画像だけモードでは，サムネイル画像だけを並べて表示する.

他に，キーワードに対する検索結果を RSS 形式で出力する機能や，登録された全 blog 記事からランダムに取り出して出力する機能がある.

検索結果の並べ方には,「スコア順」と「日付順」の 2 通りがある．スコア順では，Namazu の持つ TF-IDF をベースとしたスコアリング機能を使用している．日付順では，本文・画像抽出モジュールによって抽出された投稿時刻から，新しい時刻が一番上に来るよう（降順）に並び替えを行う.

4.2 検索モジュール

本システムでは，Namazu を用いて blog 記事の本文をインデクシングし，全文検索可能としている．分かち書きには Kakasi[12] を用いている．検索モジュールはインタフェースからクエリを受け取って Namazu の検索 cgi を実行し，検索結果をインタフェースに返す．その際，検索結果に含まれる個々の記事について，サムネイル画像のパスを調べ，検索結果に付け加える.

4.3 API の提供

本システムは，サムネイル画像や画像メタデータの取得機能を，REST 型の Web サービス「もぶろげっと API[13]」として提供している．アプリケーション開発者は，もぶろげっと API を利用することで，本システムの検索結果を容易に自分の開発したアプリケーションに取り入れることができる [3].

5 評価

本システムに対するユーザの評価を調べるため，本システムに対する意見調査を行った．得られた主な意見を表 3 に示す.

意見 1〜3 から，サムネイル画像が blog 記事の内容に対する直感的な理解を助ける役割があることが確認できる.

意見 4〜8 からは，食べ物に関する話題や趣味に関する話題などについて検索する際に，サムネイル画像がその blog 記事を読むかどうかの判断に使用される場合があると推察できる．しかし，意見 8 のように，検索対象が抽象的なものだとサムネイル画像は参考にならないという指摘もあり，サムネイル画像が有用であるかどうかは検索対象に左右される部分が大きいということが伺える.

その他，本システムに対する要望として，意見 9〜13 のようなものがあった．意見 11 は，本システムが画像を含まない blog 記事を検索対象から除外していることによる網羅性の低下を指摘したものである．これまでに本システムが収集した blog 記事を見ると，画像を含むものの割合は blog 記事全体の 10 パーセント強に過ぎない．意見 12〜13 は，本システムのクローリング性能に対する要望である．より多くの blog からより多くの記事を検索したいというユーザの要求を確認できる．また，登録される blog の URL の数が増加するに連れて，本システムのクローラはそれらを処理しきれなくなりつつあり，クローリングの効率化が必要となっている.

[11] Namazu, http://www.namazu.org/

[12] Kakasi, http://kakasi.namazu.org/

[13] もぶろげっと API, http://api.mobloget.jp/

表 3. ユーザ意見

意見 1	画像があるとその blog 記事の内容を検索結果のページだけで判断できることが多くなる
意見 2	画像で検索できると，blog 記事の内容を直感的に把握できる
意見 3	画像があるとその blog のイメージが直感的に把握できる
意見 4	生活に密着した画像が出てくるからなかなか面白い
意見 5	「鉄道」とか入れると車両の写真と旅行記が見つかったりして便利
意見 6	一番のオススメ検索語は食べ物関係です
意見 7	「ラーメン」で検索すると様々なラーメン画像と blog 記事が出てくるので店探しに使える．また，個人のブログなので遠慮ない感想が載っている
意見 8	「怒った人」,「いい雰囲気」のような，抽象的なキーワードだとサムネイル画像は役に立たない
意見 9	ソートの初期値は，日付順の方が良い
意見 10	専用 UI があるといい （メモリウムみたいなの）
意見 11	画像がない blog 記事は検索対象にならない
意見 12	巡回の対象が広くなるといい
意見 13	登録されている記事数が少ない

6 まとめと今後の課題

本論文では，画像情報を含む blog 記事検索システム「もぶろげっと」の設計，実装，評価について述べた．ユーザからの意見収集により，サムネイル画像が blog 記事の内容を判断する一助になっていることを確認した．

blog の普及と，携帯電話やデジカメの普及とが組み合わさり，個人が自分で撮った写真を容易に WWW 上で公開できる枠組みが整った．今後，blog に投稿される画像情報の総量はさらに増し，その重要性を高めて行くものと考えられる．そのような状況に際し，我々は blog に投稿される画像情報に注目し，これを大量かつ網羅的に収集・監視するシステムを開発し，一般に向けて公開した．

今後は，クローリングの効率化や，検索結果からアダルト記事を除外するフィルタリング技術の研究を行う予定である．また，画像メタデータの利用など，本システムで収集されたデータに対するより高度な検索手法を検討したいと考えている．

参考文献

[1] N. Glance, M. Hurst, and T. Tomokiyo. BlogPulse: Automated Trend Discovery for Weblogs. In *WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, 2004.

[2] 奥村学, 南野朋之, 藤木稔明, 鈴木泰裕. blog ページの自動収集と監視に基づくテキストマイニング. 人工知能学会, セマンティックウェブとオントロジー研究会, SIG-SWO-A401-01, 2004.

[3] 宵勇樹, 福地健太郎, 小池英樹. mobrium: 眺めて取り出すインタフェース. 第 4 回情報科学技術フォーラム (FIT 2005), K-040, 2005.

[4] 上原子正利, 池田貴紀, 浅井一希, 古谷楽人, 内藤裕紀. ニュース・ウェブログ記事集約サイトの開発. 電子情報通信学会論文誌, J88-D1(2):305–315, 2005.

[5] 武田英明. Weblog 研究の現状. 人工知能学会, セマンティックウェブとオントロジー研究会，SIG-SWO-A402-06, 2004.

付録 A 日付抽出のための正規表現 *

```
1: @"(?<year>\d{2,4})\s*年\s*(?<month>\d
   {1,2})\s*月\s*(?<day>\d{1,2})\s*日\D*
   (?<hour>\d{1,2})\s*時\s*(?<minite>\d{
   1,2}) 分"
2: @"(?<year>\d{2,4})\s*[-/ー－／]\s*(?<
   month>\d{1,2})\s*[-/ー－／]\s*(?<day>
   \d{1,2})\s*\D*(?<hour>\d{1,2})\s*[:：]
   \s*(?<minite>\d{1,2})"
3: @"(?<year>\d{2,4})\s*年\s*(?<month>\d
   {1,2})\s*月\s*(?<day>\d{1,2})\s*日"
4: @"(?<year>\d{2,4})\s*[-/ー－／]\s*(?<
   month>\d{1,2})\s*[-/ー－／]\s*(?<day>
   \d{1,2})"
5: @"(?<month>\d{1,2})\s*(?<day>\d{1,2})
   \s*,\s*(?<year>\d{4})"
```

※.NET Framework の正規表現

付録 B 画像メタデータの例

```
<?xml version="1.0" encoding="utf-8" ?>
<image>
    <add key="alt" value="トヨタ" />
   <add key="height" value="360" />
   <add key="post" value="フィリピンにもトヨタが進出してます．他にもホンダ，三菱も行ってます．只，走ってる車は古い年式が多い気がした．新車を買える人なんて" />
   <add key="pre" value="ンル:[旅・アウトドア］ [ 記事を作成・編集する] 天下のトヨタ 2005-03-13 12:25:37 おっぺけぺ〜な感じ " />
   <add key="type" value="jpg" />
   <add key="url" value="http://ameblo.jp/user_images/eb/ff/9e149c449561d430222dc5d857151262.jpg" />
   <add key="width" value="480" />
</image>
```