修 士 論 文

構造化アノテーション付き
電子掲示板における投稿データの
知的再利用手法に関する研究

350303185 松本 和之

名古屋大学大学院 情報科学研究科
メディア科学専攻

2004年度

目 次

第１章　はじめに **5**
1.1　電子掲示板の歴史 5
1.2　従来型電子掲示板とその問題点 6
1.3　電子掲示板タイプの様々なシステム 6
1.4　本研究の目的と構成 7

第２章　アノテーション掲示板の基本システム **9**
2.1　アノテーション掲示板システムの実装 9
2.2　アノテーション掲示板システムの運用 11
2.3　アカウントと認証 13
2.4　投稿とスレッドの作成 14
2.5　投稿データの形式 16

第３章　投稿クラスを用いた構造化アノテーション **19**
3.1　掲示板の投稿データに対するアノテーションの必要性 19
3.2　投稿クラスと構造化アノテーション 21
3.3　投稿クラスの定義 22
3.3.1　投稿クラスの継承 22
3.3.2　投稿クラスの定義と投稿フォームの生成 23
3.3.3　プロパティのオーバーライド 27

第４章　投稿の絞り込み検索 **29**
4.1　絞り込み検索とその手順 29
4.2　検索対象の選択 29
4.3　絞り込み条件の設定 31
4.4　絞り込み検索結果の表示 33
4.5　絞り込みルーチンのユーザ定義 34
4.6　複数クラスにまたがった絞り込み検索 36

第5章　投稿データに対するオペレーション　39
5.1　オペレーションとその手順 39
5.2　オペレーション対象の選択 39
5.3　3通りのオペレーション 40
5.3.1　方法1：他のユーザが作成したオペレーションの利用 41
5.3.2　方法2：Perlスクリプトによるオペレーションの作成 42
5.3.3　サブルーチンの定義 46
5.3.4　方法3:APIやサブルーチンの組み合わせによるオペレーション 49

第6章　オペレーション結果を用いた知識発見の支援　51
6.1　オペレーション結果の投稿 51
6.2　オペレーション自身の投稿 52
6.3　オペレーションデーモン 53

第7章　関連研究　55
7.1　TelMeA 55
7.2　機械学習による電子掲示板からの評判情報抽出 56

第8章　今後の課題　57
8.1　オペレーション結果表示インタフェースの拡張 57
8.2　スクリプトのエラー処理とセキュリティ 58
8.3　複数掲示板でのデータ共有 59

第9章　おわりに　61

参考文献　63

付録　65

謝辞　127

概要

インターネットの歴史が始まる前から、電子掲示板というコミュニティは多くの人々を惹きつけてきた。現在も様々な電子掲示板システムが存在し、数多の投稿を通じて様々な情報のやりとりが日常的に行われている。それらの情報の中には、我々にとって有用な情報も大量に含まれていることから、掲示板システムは知識の収集・蓄積システムとみなすことができる。まさに知識の宝庫と言えるであろう。しかし、従来の電子掲示板システムでは投稿がプレーンテキストであるため、単純なテキストマッチングによる検索程度しかできず、投稿データの再利用性は極めて低い。本論文では、構造化アノテーションを投稿に対して付与することによって、検索を始めとする様々な高度な処理を効率的に行うことのできる、投稿データの再利用性が高い電子掲示板システムについて述べる。

ユーザは、投稿クラスと呼ばれるデータ構造を定義することによって投稿に対する構造を自由に定義することができ、その定義から自動生成される投稿フォームに入力するだけで、構造化アノテーションが自動的に付与された投稿を行うことができる。蓄積された投稿データは、各ユーザの目的に応じた利用が可能となる。投稿データの絞り込み検索は、システムで用意される GUI や Perl プログラムを用いてユーザ定義したサブルーチンによって自由な条件を設定することが可能である。また、投稿データを処理するオペレーションはその自由度や難易度の異なる 3 通りの方法を用意することにより、幅広いユーザ層に対応している。

本研究では、投稿構造の自由な設定を許すことで用途が限定されず、また投稿データの利用方法も限定されない新しい電子掲示板システムの仕組みを構築した。

Abstract

Online communication systems or bulletin board systems have been widely used before the age of the Internet began. Such communication systems accumulate human knowledge as messages submitted by the users. Since most data included in the systems are just plain texts, it is difficult to reuse data just by employing simple pattern matching techniques. In this paper, we present a new online communication system that contains highly-reusable semantically-annotated messages. Semantic annotation on messages is realized by user definable data structures called message classes. The message class is defined using XML (eXtensible Markup Language) and has functionality to generate an appropriate message input form. The users can write messages on the form, and then semantically-annotated (structured) messages are automatically generated. Our online communication system can also manipulate the accumulated message data to search for some valuable information and to generate mined results by applying some operations to whole or some part of the messages. For example, the system can generate a personalized ranking of a user's preferred restaurants based on reviews included in the messages. To flexibly manipulate the message data, our system has a mechanism for filtering the messages and for applying operations to the filtered messages. Such operations are implemented by integrating user-customizable operators and Perl programs. Therefore our system can not only accumulate human knowledge as structured message data but also help the users to discover useful information by filtering and mining the messages.

第1章　はじめに

1.1　電子掲示板の歴史

現在では、インターネットを通じて様々なサイトで電子掲示板を目にすることができるが、日本での電子掲示板の歴史の始まりは、1980年代、NIFTY SERVEやPC-VANなどの商用パソコン通信サービス時代にまで遡る。当時、パソコン通信の標準サービスとして電子掲示板が存在しており、様々な情報がやりとりされていたが、ハンドルネームを変えてもIDやhostによって個人が特定できてしまうため、匿名性は低かった。1991年、アメリカではインターネット協会が設立され、日本でも翌年IIJ [9]が設立されたことを契機に、インターネットが商用サービスにも解放されることになった。実際に日本のISPがインターネット接続サービスを開始した時期は1995年ごろであり、1995年はインターネット元年と呼ばれている。以後、インターネットにおけるCGIを利用した書き込みやすい電子掲示板やチャットによって、人々の関心はパソコン通信からインターネットに移行していった。

「miniBBS」という電子掲示板に代表される当初の電子掲示板では、投稿者名、メールアドレス、タイトル、本文を書き込める程度で、また投稿が時間順に表示されるだけの非常にシンプルなものであった。次第にminiBBSを元にした様々な掲示板が乱立するようになった。まず、投稿に対して一行レスが付けられるタイプの掲示板が現れるようになり、さらにレスをした投稿が最上部に表示されるタイプの掲示板も増えた。次に1行レスではなく通常の投稿のように長文のレスを付けられるタイプの「マルチBBSII」「HyperBBS」という掲示板などが現れた。このタイプの掲示板の出現により、掲示板には話題の単位である「スレッド」が並ぶようになる。すなわち、各スレッドごとにminiBBSを内包しているような形態である。掲示板の多くは個人ホームページに掲示板を設置する程度だったが、「あめぞう」と名乗る人物が掲示板を話題ごとに並べる「掲示板コミュニティ」的な使い方を考案したことにより、現在の「2ちゃんねる」[21]のような形態の掲示板

文化が発展していった。

このように電子掲示板はパソコン通信時代からネットワークを利用した基本的な情報交換手段を担ってきたとともに、インターネットの普及を支えてきたと言っても良いであろう。しかし、電子掲示板の投稿一つ一つを見たとき、そのほとんどがプレーンテキストのみから構成されていることは昔から不変である。

1.2 従来型電子掲示板とその問題点

電子掲示板は前節で述べたように歴史が長く、様々な仕組みのものが提案・利用されてきた。検索エンジンを用いて調べ物をするとき、電子掲示板のログが検索結果としてヒットすることも多い。このことからも電子掲示板は、投稿データに多くの情報が集約されている、知識の宝庫であると言えるだろう。しかし、電子掲示板に蓄積された知識を抽出しようとするときに、従来のシステムでは単純なテキストマッチングによる検索程度しかできず、知識を抽出する効率は良いとは言えない。たとえば、ある駅付近にある飲食店に関する情報を投稿する掲示板(またはスレッド)があったとする。その駅付近で食事をしたいユーザがいて「予算が1000円以下で、駅から徒歩5分以内でいける中華料理店」を知りたいと思っているとき、「中華」などのキーワードを用いて検索を行ったとしても、予算を表した部分がどこであるかという情報や、駅から徒歩でどれくらいかかるかという情報を投稿本文から抽出することは容易ではないため、順に投稿を読み進める以外の方法でその要求を満たすことは難しい。

1.3 電子掲示板タイプの様々なシステム

プレーンテキストよりも構造化された情報を投稿できる、電子掲示板タイプのシステムもいくつか存在する。そのいずれにおいても、特定の目的に限定した掲示板となっている。

1999年春に開設された「自動アンケート作成」は、その名の通りアンケートに特化した掲示板タイプのシステムである。任意のユーザによってアンケートのテーマが投稿されると、すべてのユーザがアンケートの回答を行えるようになる。アンケートは既存の項目から選択するか、新規に項目を作成することができる。アンケート結果は随時、投票数や割合、棒グラフなどにより表示される。この「自

動アンケート作成」は2002年5月に閉鎖されたが、現在でもアンケートに特化した「オリジナル☆ランキング」[24] などのシステムが残っている。

「add your own」[1] は地域ごとのレストランの情報を書き込む掲示板タイプのシステムである。書き込む情報としては、店名・ジャンル・平均予算・住所・電話番号・コメントがあるが、蓄積された情報を閲覧する以外に投稿データの利用法は提供されていない。また、1つの店舗に関する情報は常に1つしかなく、既にある店舗情報を投稿するには、既存の投稿を「編集」することしかできない。

「価格.com」[23] は様々な商品の価格を比較するサイトであり、商品を軸とした多種多様な情報を利用できる。その中でも、このサイトにおいて最も重要な意味を持つ情報が「価格」情報であり、商品ごとに安値ランキングが表示される。価格情報の収集は、契約している店が専用のフォームから送信することにより、サイトにおける価格情報に反映されるようになっている。また、各商品ごとに「口コミ掲示板」などによる口コミ情報があり、価格・デザイン・性能など、商品の種類ごとに用意された視点から「良い」「悪い」の2択による評価フォームとその結果が表示される。また各商品についてユーザが話し合える掲示板が商品ごとに用意されている。性能などの商品に関する情報や、商品を評価する際の指標は、管理者側によって決定され、投稿データの利用についても、管理者によって用意された方法に限られる。

以上のように、ある特定の目的に限定された掲示板タイプのシステムはいくつか存在するが、その投稿データの構造や利用法は管理者などによってあらかじめ決められたものに制限されてしまう。

また、大量のテキストデータから何からの知識を発見するためのより一般的なアプローチとして、テキストマイニングと呼ばれる研究分野がある [26]。しかし、テキストマイニングは一般に構造を持たない任意のテキストデータを対象とするため、精度の点で問題があり、ここで我々が目指しているような、人々の持つ断片的な知識をかき集めて、個人にとって意味のある結果を導き出す、という目的には適用できない。

1.4 本研究の目的と構成

本研究では、電子掲示板の投稿データの再利用性を高めるために、プレーンテキストによる投稿ではなく、投稿者が投稿に対して「構造化アノテーション」を

付与し、構造化アノテーション付きの投稿データを用いることによって検索を始めとする様々な高度な処理を効率的に行えるようにする。このようにして拡張された電子掲示板を「アノテーション掲示板」と呼ぶ。アノテーション掲示板では、投稿データの利用方法がシステムから提供されたものに限定されず、ユーザの求める様々な処理を掲示板システム内で自由に行えるシステムの実現を目指す。

本論文の構成は以下のようになっている。

2章では、アノテーション掲示板の基本システム部について述べる。3章では、本研究の核となる、構造化アノテーションと投稿クラスに関して述べる。4章では、蓄積された投稿データの絞り込み検索について述べ、5章では、投稿データの高度な利用法であるオペレーションについて述べる。6章では、オペレーションを用いた様々な知識発見の支援機能について述べる。7章で関連研究について触れ、8章と9章でそれぞれ今後の課題とまとめを述べる。

第2章　アノテーション掲示板の基本システム

2.1　アノテーション掲示板システムの実装

本研究で構築したアノテーション掲示板システムは、図 2.1 に示すような構成になっている。本システムは、幅広いプラットフォームに対応するために完全にWebベースとなっている。したがって、特別なクライアントやプラグインを必要とすることなく、Webブラウザのみですべての機能を利用することが可能である。

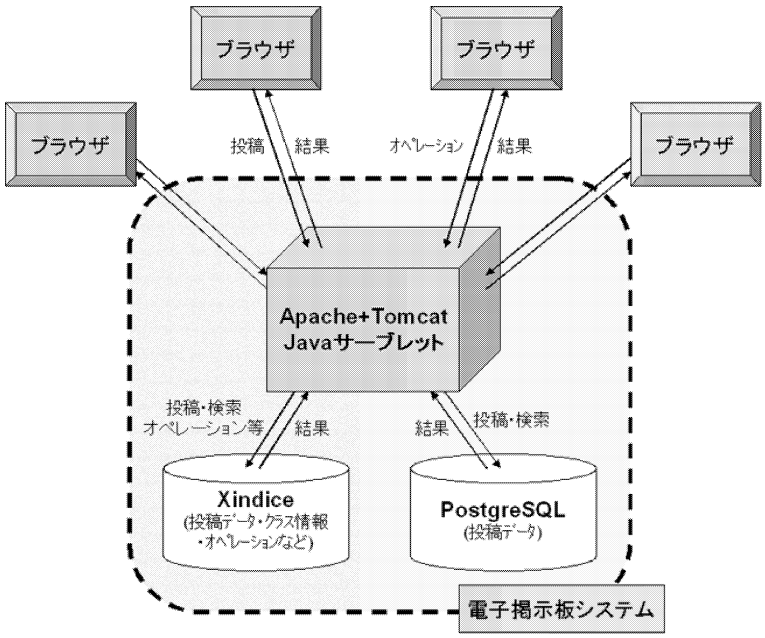

図 2.1: 本システムのシステム構成

サーバサイドは、すべてJavaサーブレット [16] として実装されており、サーバ・クライアントの通信部分に関して簡素に記述できる上、プラットフォームの変化

にも強い。また、豊富なAPIが利用でき、生産性が高いと言えるだろう。なおアプリケーション・サーバとしてはTomcat [2]を、Apache [4]のプラグインとしてWindows XP Professional上で利用している。

一部の設定ファイルを除いた、投稿データなどのほとんどのデータはXML (eXtensible Markup Language) [18]で記述されている。XMLという標準化されたフォーマットを利用することにより、プラットフォームが限定されず、データの取り扱いも容易となる。もちろん本システムで使用しているJavaにおいても、Xerces [7]やXalan [6]といったXMLの操作をするためのパーサや変換ツール等、非常に多くのAPIやライブラリを利用できる。また、他のプラットフォームとのデータ交換性にも優れている。

また、XMLは拡張性の高い言語であり、構造単位での自由な拡張が可能である。詳しくは3.2節で説明するが、本システムでは投稿データの形式をそれぞれの投稿ごとに変更できる。しかしXMLを利用しているため、投稿ごとにデータ形式が異なっていても、特に問題なくプログラムによって処理することできる。また今後、投稿データの一部分に対してアノテーションを付加するなど、投稿データの拡張がされたとしても、基本システムをほとんど変更することなく運用を続けることが可能である。

XMLデータを管理するデータベースとしてXindice [5]を利用している。Xindiceは、Apache Software Foundation [3]のプロジェクトの1つとして開発されているネイティブXMLデータベースで、すべてJavaで実装されており、Java用のAPIも用意されているため、本システムとの親和性は非常に高い。また、ネイティブXMLデータベースであるため、リレーショナル・データベースとは異なり、XMLに特化した内部構造を持っており、XML文書内のデータをたどる処理等も効率良く行える。

Xindiceに保存されたXML文書を、クライアントであるブラウザに出力する際、XML文書に対してスタイルシートを適用して、HTML形式に変換している。スタイルシートであるXSL [20]文書はXML形式で記述されているので、これもXindiceに保存されている。このため、スタイルシートを書き換えるだけで電子掲示板の見栄えを変更することができる。実際、管理者権限でログインする管理者モードでは電子掲示板の名称や見栄えの変更を行うことができる (3.4節参照)。

ところで、XMLの欠点の1つとして、検索が遅いことが挙げられる。大量の投稿データがあり、キーワードを用いて全文検索しようと思ったとき、全てのXML文書を検索するのには多くの時間的コストがかかってしまう。そこで、単純なキー

ワードマッチングによる全文検索を行うときに限っては、リレーショナル・データベースである PostgreSQL [14] を利用することによって、検索の高速化を行っている。

2.2 アノテーション掲示板システムの運用

本掲示板には、「投稿」という概念に加え、話題の単位である「スレッド」、スレッドより話題の単位の大きい「掲示板カテゴリ」という 3 段階の単位がある。投稿やスレッドの作成はすべてのユーザが自由に行えるが、掲示板カテゴリは管理者のみが作成するものとした。これは、本システムにおける最も大きな話題の単位である掲示板カテゴリが自由に作成できると、掲示板カテゴリが乱立することが懸念され、投稿者やスレッド作成者を適切な掲示板カテゴリに導くことが困難になると考えたからである。図 2.2 のように、左フレームに掲示板一覧、右上フレームに選択されている掲示板カテゴリに属するスレッド一覧、さらに右下フレームには選択されているスレッドの投稿一覧がそれぞれ表示される。これが本システムの基本画面である。

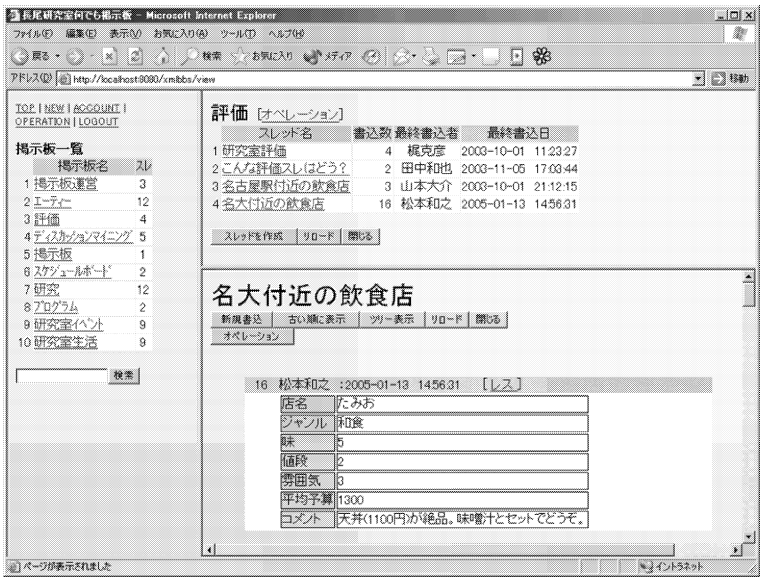

図 2.2: システムの基本画面

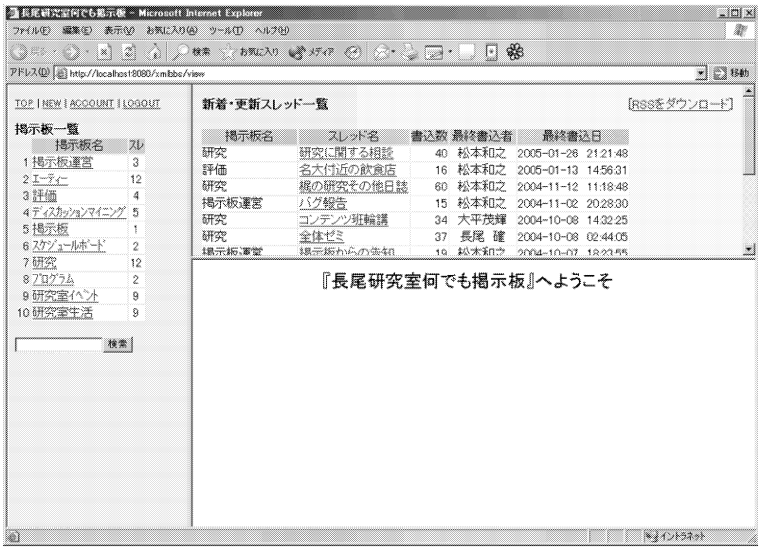

図 2.3: 更新のあったスレッド順に並んだスレッド一覧 (右上フレーム)

また、本システムにアクセスすると最初は図 2.3 に示したように、右上フレームに更新のあったスレッドが更新順に並べられた状態になっているため、容易に新しい投稿をチェックすることができる。また、投稿があるたびに RSS (RDF Site Summary) [15] を更新しており、RSS ファイルにはアクセス制限がかかっていないため、本システムに直接アクセスしなくても、RSS リーダ等を利用することによって更新の有無を確認できる。図 2.4 に示すように、この RSS には、更新のあった掲示板名・スレッド名・投稿者・投稿日時・投稿タイトル・URL が含まれている。

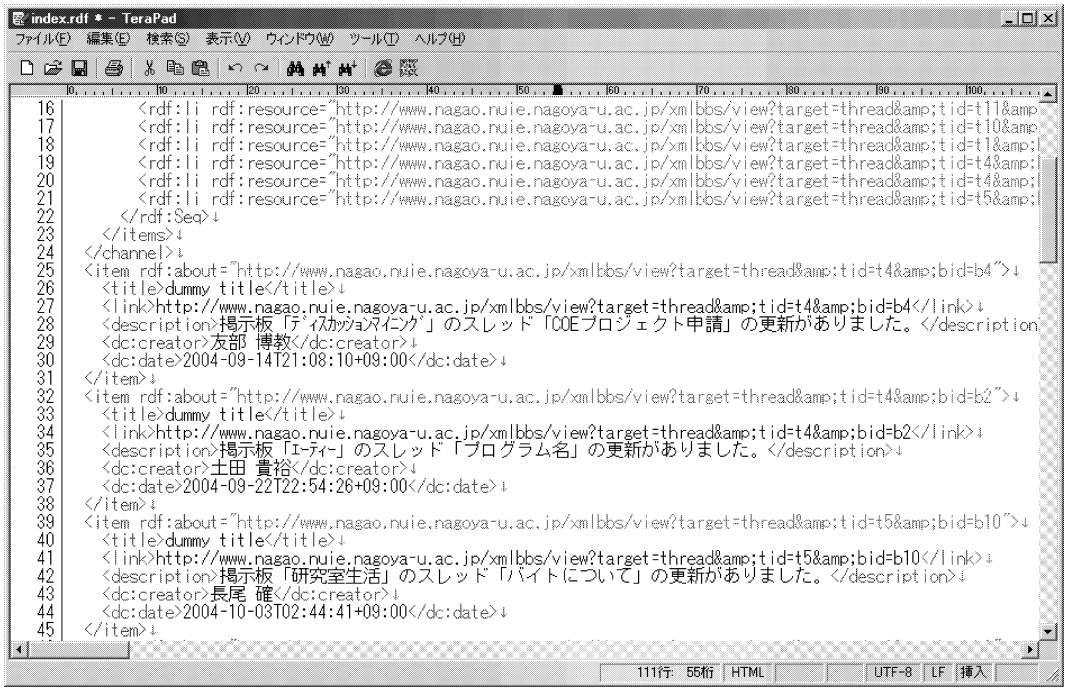

図 2.4: 出力する RSS ファイルの一部

2.3 アカウントと認証

本掲示板を閲覧したり投稿したりするにはシステムへのログインが必要となる。そのためにはまずユーザアカウントを作成しなければならない。ユーザアカウントは、ユーザ ID・パスワード・フルネームの 3 項目を入力することで作成される。ユーザ ID とパスワードは、本システムにログインするときに入力するもので、フルネームは掲示板の投稿に表示される名前である。

通常のユーザアカウントとは別に、管理者アカウントも存在する。管理者アカウントでログインすると、「ユーザアカウントの管理」「掲示板システムの設定」「掲示板の操作」が行える。「ユーザアカウントの管理」では、登録されているすべてのユーザアカウントを操作できる。具体的には、ユーザ ID・パスワード・フルネームの変更ができるほか、ユーザアカウントを削除したり、新規に作成することもできる。「掲示板システムの設定」では、掲示板システム名と掲示板の配色の設定ができる。「掲示板の操作」では、掲示板カテゴリの作成や変更ができる。また、既存の投稿を削除したり変更したりすることも、管理者権限でしかできない。

2.4　投稿とスレッドの作成

スレッドを新たに作成するには、まずどの掲示板カテゴリに作成するのかを決定する。左部フレームから、スレッドを作成したい掲示板カテゴリをクリックして、右上フレームにスレッド一覧を表示させる。スレッド一覧の最下部に「スレッド作成」ボタンがあるので、それをクリックすると、図2.5のようなフォームが現れる。このフォームに、スレッド名を入力し、さらにそのスレッドにおける1番目の投稿を作成する。本掲示板システムでは、スレッドを作成すると同時に、そのスレッドにおけるガイドラインとなるような投稿を行い、他のユーザはその投稿を読んでスレッドの方向性等を確認した上で投稿する、という形になる。

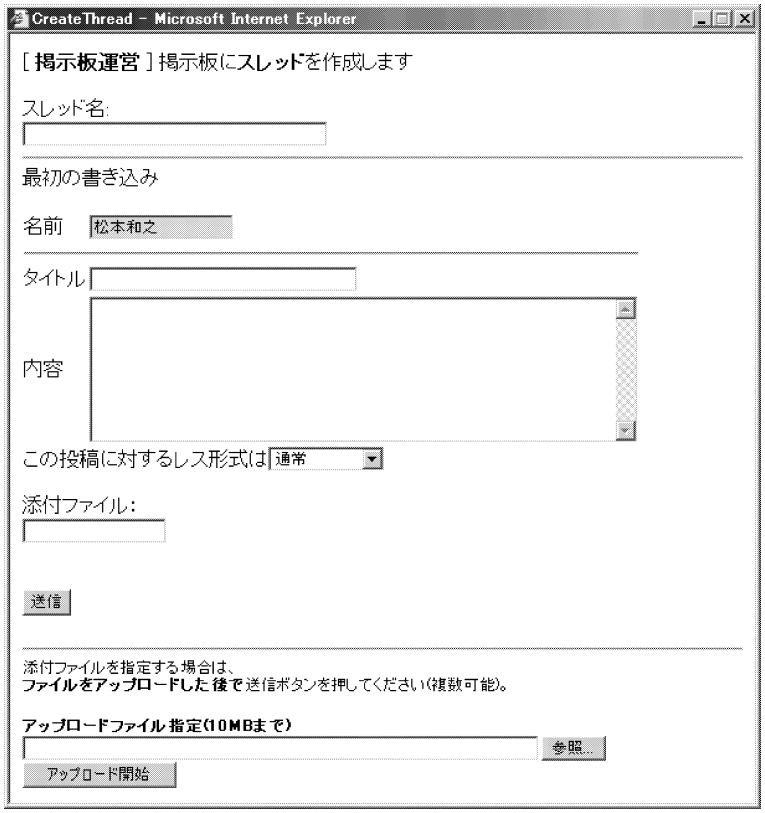

図 2.5: スレッド作成フォーム

投稿を行うには、投稿をしたいスレッドをクリックして、右下フレームにそのスレッドに投稿されている投稿一覧を表示させる。このフレームの最上部と最下部に「新規投稿」ボタンがあるので、これをクリックすることにより、図2.6のよう

な投稿フォームが現れる。この投稿フォームにおいて、投稿者名・レス対象・(投稿) 形式に関する情報はすべて自動で入力されており、変更もできないようになっているので、なりすまし等の行為はできない。掲示板システムにログインするときにセッションを作成し、そのセッションに対してユーザ ID をセットしているので、投稿フォームを表示する際に、データベースからそのユーザ ID を持ったユーザを探し出すことによって、投稿者の名前を取得できる。

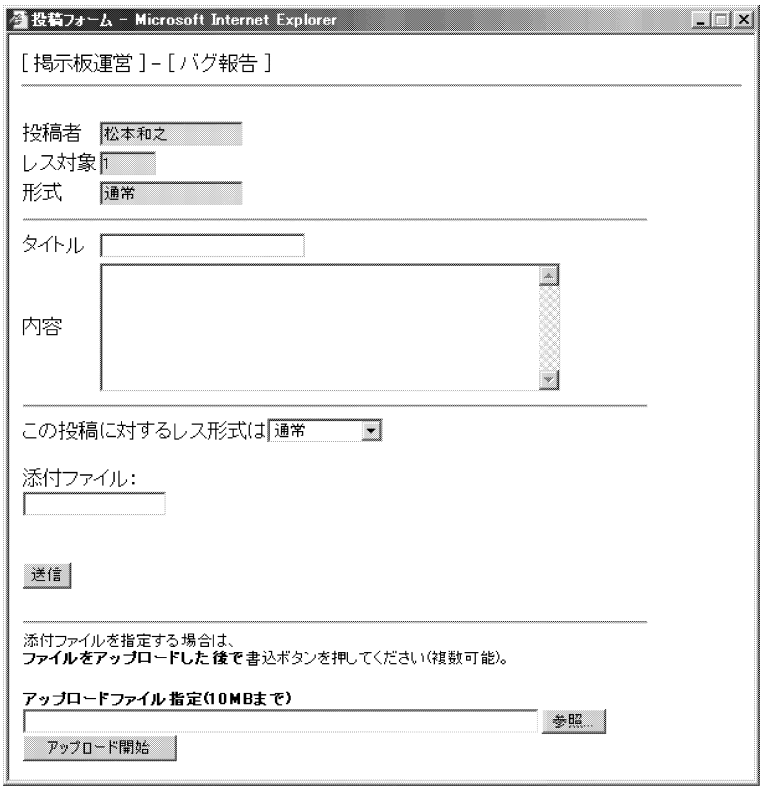

図 2.6: 投稿フォーム

投稿形式は投稿クラスによって異なるが、ここではデフォルトの投稿クラスの投稿について説明すると、図 2.6 のようにタイトルと本文を入力する部分があるのでそれぞれ記述し、「この投稿に対するレス形式は」のすぐ右にあるリストボックスから、「この投稿に対して返信する人に、どのような形式で返信して欲しいか」を選択する。この場合の「形式」は、「投稿クラス」と呼ばれるものであり、次章で詳しく説明する。

投稿に対して、添付ファイルを指定することもできる。本システムでは、添付

ファイルの数や種類に制限はない。添付ファイルのサイズに関しては制限をすることも可能である。

2.5　投稿データの形式

サーブレットが受け取った投稿データは、以下に示すような形式でデータベースに保存される。なお、ここに示したXMLデータは、1投稿当たりの投稿データである。

```
<item id="m13" writer="matsumoto">
  <date>2005-01-13</date>
  <time>12:37:08</time>
  <res>9</res>
  <restype>ひとこと</restype>
  <class>通常</class>
  <host>192.168.xxx.xxx</host>
  <text>
    <form>
      <input article="タイトル">賛成です</input>
      <input article="内容">
        それは良い案だと思います。
        <br />
        今週中に実装したいと思います。
        <br />
        良い意見をありがとうございました。
      </input>
    </form>
  </text>
</item>
```

各投稿には投稿 ID(item 要素の id 属性) が付けられるほか、投稿者 ID(item 要素の writer 属性)、日付 (date)、時間 (time)、返信先投稿 (res)、返信形式 (restype)、投稿クラス (class)、ホスト情報 (host)、本文 (text) が記録される。本文は、構造化アノテーションされているので form 要素や input 要素などによって構造化された状態で保存される。

第3章　投稿クラスを用いた構造化アノテーション

3.1　掲示板の投稿データに対するアノテーションの必要性

本研究は、掲示板に対するアノテーションを用いることによって、従来の電子掲示板の投稿データに不足している情報を補う、というのが基本的な考え方である。ここで、アノテーションとは、簡単に言うと、オリジナルのコンテンツに対して関連付けられるメタ情報一般のことであり、これによって情報の価値を高めることができる。

世の中には非常に多くのコンテンツが存在する。Web 上に存在するものだけ取り上げても、音楽・動画・静止画・日記・掲示板など、多種多様なものが存在する。これらのコンテンツは、非常に高い知識的価値を持っているが、オリジナルコンテンツのままでは、高度な利用が困難である。そこで、このオリジナルコンテンツに対して、図 3.1 のようにアノテーションを施すことによって、検索・要約・知識抽出などの高度で様々な利用を可能にすることを目指す。

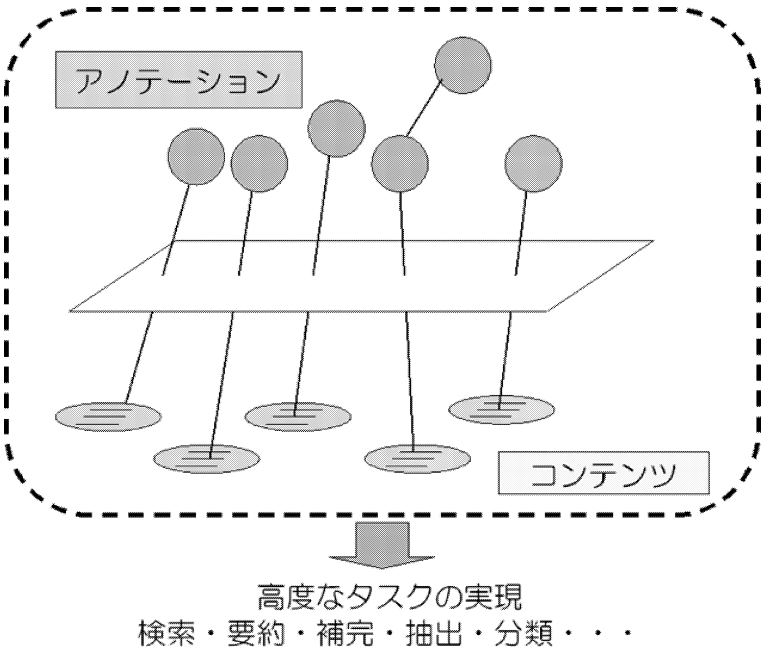

図 3.1: アノテーション

本研究では、電子掲示板の投稿に対してアノテーションを付与することによって、投稿データの高度利用を目指す。電子掲示板の投稿データは一般的にプレーンテキストとして表現される。例えば「名古屋駅の近くにある○○という中華料理のお店に行ってきて、全品安くておいしかったが、店の雰囲気はあまり良くなかった」といった趣旨の投稿があるとする。この投稿には、店に関する情報が書かれていて、店名・ジャンル・値段・味・雰囲気などの情報が含まれている。しかし、これらの情報を機械処理によって投稿から抽出するのは容易なことではない。これは、どの部分が店名に関する情報で、どの部分が値段に関する情報か、といった情報を投稿者がメタ情報として付与、すなわちアノテーションを作成することによって、機械処理しやすくなり、情報の価値を大幅に高めることができる。

特に、電子掲示板というコミュニティでは、時間とともに投稿が増加し続ける。このように続々と蓄積されていくデータ全体から、あるまとまった知識が抽出できることの意義は大きいと考えられる。

このように、電子掲示板の投稿に対してアノテーションを付与することによって不足している情報を補い、情報の価値を高めることができる。

3.2 投稿クラスと構造化アノテーション

従来の電子掲示板システムにおける投稿データには、テキストのどの部分がどのような意味を持っているかという情報が欠如している。そこで、まずは投稿に対して構造を与えることから始める。

投稿者は、投稿に対して与える構造を定義するデータである「投稿クラス」を定義して、自分の投稿に対する返信としてどのような構造を持った投稿が欲しいのかを規定する。例えば、投稿クラスの一つである「評価.飲食店」クラスが持つプロパティには「店名」「ジャンル」「平均予算」「味の評価値」などがある。求める返信形式として「評価.飲食店」を指定した投稿に対してユーザが返信するために、その投稿の横に表示された「返信」と書かれたリンクをクリックすると、この投稿クラスの定義から自動生成された、図3.2で示すような投稿フォームが現れるので、各入力フィールドを埋めることにより容易に構造化された投稿が可能となる。このように、投稿内容に対してプロパティを対応させることにより、構造化された投稿を行う本研究の仕組みを、「構造化アノテーション」と呼ぶ。

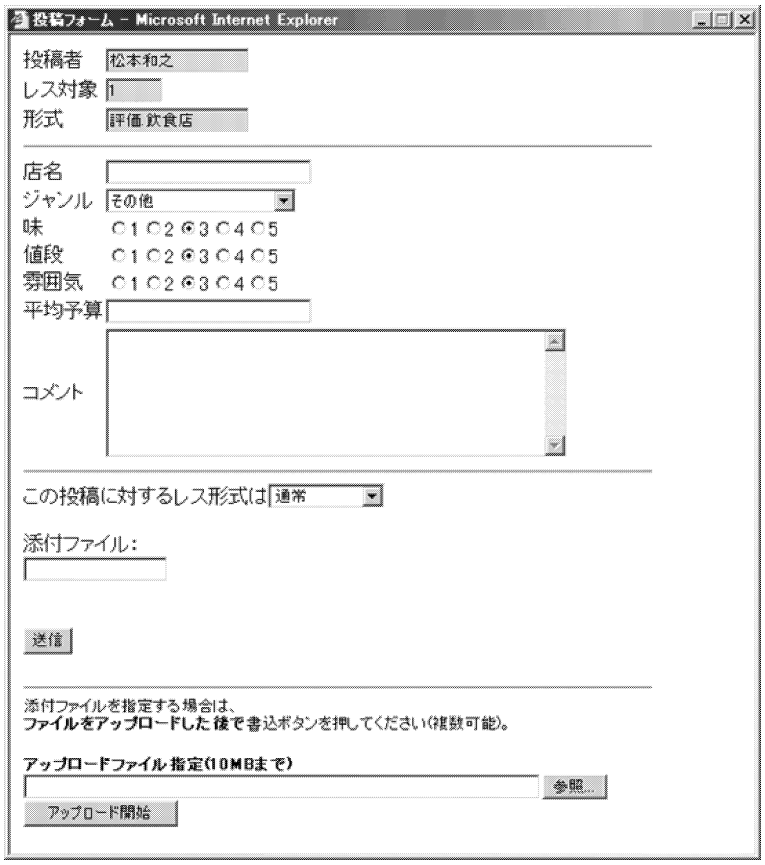

図 3.2: 評価.飲食店クラスの投稿フォーム

3.3 投稿クラスの定義

投稿クラスは、投稿クラス名・親投稿クラス名・投稿構造・利用可能なオペレーション・利用可能なサブルーチン・プロパティのオーバーライドといった情報によって定義される。

3.3.1 投稿クラスの継承

すべての投稿クラスはデフォルトの投稿クラスから継承されたクラスとして捉えられる。本システムにおける継承とは、オブジェクト指向におけるそれと同様で、上位クラスの性質や機能を受け継ぐことである。本システムにおいては、投

稿クラスを継承することによって、親クラスのプロパティを利用したり、親クラスのオペレーションやサブルーチンを利用したりすることができる。

クラス名は、クラス間の階層関係が分かるように「.(ドット) 記法」を用いて表現する。例えば飲食店の評価についての投稿構造を持った「評価. 飲食店」クラスは、(飲食店に限らず) 様々なものについて評価を行う投稿構造を持った「評価」クラスを継承したクラスである。ただし、デフォルトの投稿クラスについてはクラス名に含めないことにする。どの投稿クラス名にもデフォルトの投稿クラスが含まれるのは冗長だからである。なお、デフォルトの投稿クラスは「タイトル」「本文」の 2 つのプロパティのみから成る投稿構造を持ち、スレッドの作成時と同時に行う投稿は、このデフォルトの投稿クラスになる。

3.3.2 投稿クラスの定義と投稿フォームの生成

投稿クラスの定義は、「RELAX NG」[12] を拡張した書式に従って記述されている。RELAX NG は、XML で記述されたスキーマ言語であり、「RELAX Core」[11] と「TREX」[10] を元にして作られた。2001 年に、OASIS [13] によって標準化されている。XML におけるスキーマ言語とは、要素や属性の配列に関して正しい並び方を明確に定義したものであり、配送途中で壊れたり、何らかの理由で間違った XML 文書が送信された場合でも、受信側で人間の手を介さずにチェックすることが可能になる。

ここで、RELAX NG の簡単な例を示す。以下のような XML 文書があったとき、

```
<addressBook>
  <card>
    <name>Kazuyuki Matsumoto</name>
    <email>matsumoto@nagao.nuie.nagoya-u.ac.jp</email>
  </card>
  <card>
    <name>Daisuke Yamamoto</name>
    <email>yamamoto@nagao.nuie.nagoya-u.ac.jp</email>
  </card>
```

```
</addressBook>
```

RELAX NG では以下のようなスキーマとなる。

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/0.9">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <data type="string"/>
      </element>
      <element name="email">
        <data type="string"/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

すなわち、XML 文書における要素 E を記述すべき位置に、element 要素を記述し、その name 属性として E の要素名を記述するというのが基本的なルールであり、この例を見れば分かるように、RELAX NG は非常にシンプルな構文の XML で記述される。さらに、Relaxer [8] などの非常に強力なデータバインディングツールを利用できるため、投稿クラスの定義は RELAX NG に従って記述することにした。

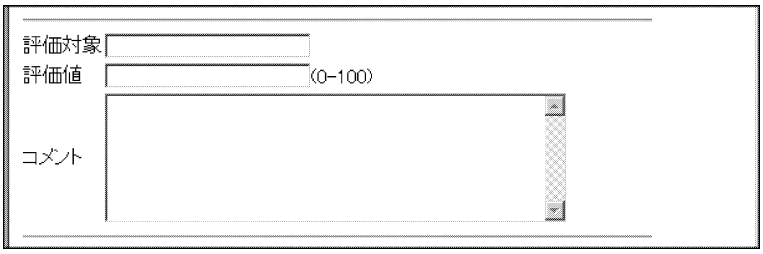

図 3.3: 「評価」クラスの投稿構造

本研究においては、図 3.3 に示すような投稿構造を持った投稿構造を定義するためには、投稿データ (投稿内容を表すノードを抜粋) が例えば以下のようになるので、

```
<form>
  <input article="評価対象">ABC ショップ</input>
  <input article="評価値">84</input>
  <input article="コメント">店員の対応があまり良くないと思う。</input>
</form>
```

このスキーマは以下のように定義される。

```
<?xml version="1.0" encoding="x-sjis-unicode" ?>
<element name="form" xmlns:bbs="http://www.nagao.nuie.nagoya-u.
 ac.jp/xmlbbs/">
  <element name="input">
    <attribute name="article">
    <value>評価対象</value>
    </attribute>
```

```
    <data datatypeLibrary="http://www.w3.org/2001/XMLSchema-dat
     atypes" type="string" />
    <bbs:component count="1" type="text" />
  </element>
  <element name="input">
    <attribute name="article">
    <value>評価値</value>
    </attribute>
    <data datatypeLibrary="http://www.w3.org/2001/XMLSchema-dat
     atypes" type="integer">
    <param name="minInclusive">0</param>
    <param name="maxInclusive">100</param>
    </data>
    <bbs:component count="2" type="text" />
  </element>
  <optional>
    <element name="input">
      <attribute name="article">
      <value>コメント</value>
      </attribute>
      <data datatypeLibrary="http://www.w3.org/2001/XMLSchema-d
       atatypes" type="string" />
      <bbs:component count="3" type="textarea" />
    </element>
  </optional>
</element>
```

プロパティのうち、店名は自由に記述できる文字列であるべきであり、ジャンルは選択式であることが望ましい。味・値段・雰囲気の評価値も、何段階かの選択式であって欲しいだろう。平均予算は自由に記述できた方が良いが、数値で入力すべきである。コメントは書かなくても投稿できるようにしたほうが良いかも

しれない。このように投稿構造には、値の型や入力方式、オプショナルであるかといった情報を含められることが望ましい。しかしRELAX NGは、XML文書における構造やその内容について記述するものであるため、値の型や、オプショナルであるかどうかという情報は記述できるが、入力方式に関しては定義できない。

そこで本研究ではRELAX NGを拡張し、入力方式に関しても定義できるようにした。例えば、先に示したスキーマ中に合計3ヵ所あるbbs:component要素のtype属性がこれに当たる。評価対象と評価値に関してはtype=”text”となっており、テキストボックスによる入力であることを示しているが、コメントに関してはtype=”textarea”となっているため、テキストエリアによる入力となる。しかし、RELAX NGを使う利点はRelaxerなどの便利なツールを利用できることにもあるため、RELAX NGを拡張した要素に関しては、名前空間(ここでは「bbs」という名称)を利用することによって容易に区別できるようにした。すなわち、容易にRELAX NGに準拠した形式に変換できるということである。

以上のように掲示板における投稿構造のスキーマを定義することによって、投稿構造を規定すると同時に、それから適切な投稿フォームを自動生成できる。

投稿フォームを埋めることによって作成された、構造化アノテーション付きの投稿が送信されると、システムは受け取った投稿の妥当性をチェックする。まず、データベースに問い合わせ、受け取った投稿クラス名からその投稿クラスの投稿構造を取得する。投稿構造に含まれるデータ型の情報や必須項目であるかどうかという情報を用いて、受け取った投稿に不備はないかを確認する。ここでもし不備が発見された場合はエラーメッセージを出力し、再び投稿の作成を促す。不備が発見されなかった場合は、ここでデータベースに投稿内容が保存される。

3.3.3 プロパティのオーバーライド

投稿クラスを継承するとき、プロパティの過不足が生じる。このとき、上位クラスのプロパティ値を、下位クラスのプロパティ値を用いて表す必要がある。

ここでは「評価」クラスと、その下位クラスである「評価.飲食店」を例にとって説明する。「評価」クラスには「評価値」というプロパティと「評価対象」というプロパティが存在する。「評価値」プロパティは0,⋯,100の整数値をとる。また「評価.飲食店」クラスには0,⋯,5の整数値をとる「味の評価値」「値段の評価値」「雰囲気の評価値」の3つの評価値が存在し、これは上位クラスの「評価値」プロパティと深い関係があると考えられる。ここで、「評価」クラスの「評価値」をA、

「評価.飲食店」クラスの「味の評価値」,「値段の評価値」,「雰囲気の評価値」をそれぞれa, b, cとおくと、

$$A = (a + b + c) * 100/15$$

とすると、上位クラスと下位クラスの評価値の関係が表せると考えられる。このように、下位クラスに属するプロパティ値を用いて、上位クラスに属するプロパティ値を決定することを、「プロパティのオーバーライド」と呼ぶ。

「評価」クラスの「評価対象」プロパティはstring型をとる。また「評価.飲食店」クラスには「店名」というプロパティが存在し、これら2つのプロパティ間には深い関係があると考えられる。ここで、「評価.飲食店」クラスにおける「店名」プロパティ(これをdとおく)は、飲食店の評価対象と見なすことができるので、「評価」クラスにおける「評価対象」プロパティ(これをDとおく)と意味的に等しいと言えるだろう。よってこの場合、

$$D = d$$

という式によってプロパティのオーバーライドが行われる。

第4章　投稿の絞り込み検索

蓄積された投稿は、構造化アノテーションを用いて絞り込み検索を行うことができる。すなわち、図 4.1 のように各プロパティが持つ値を利用した、高度な絞り込み条件を指定できる。例えば、「飲食店評価」クラスの投稿で「平均予算が 1000 円以下の中華の店」といったような条件で投稿を絞り込むことができる。

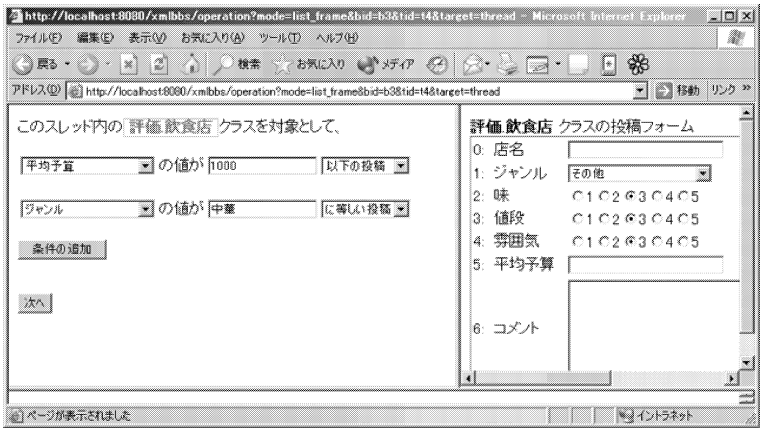

図 4.1: 絞り込み条件の設定

4.1　絞り込み検索とその手順

絞り込み検索の手順は、最初に検索対象を決定し、次に絞り込み条件を設定し、最後に結果の表示順序を指定する。このようにして、実際に絞り込み検索結果の表示が行われる。

4.2　検索対象の選択

検索対象の指定には二段階ある。

まず第一段階として、掲示板システム全体や、掲示板カテゴリ、スレッドといった対象を選択する。これは掲示板の画面においてクリックするボタンまたはハイパーリンクによって区別する。図 4.2 中に 1 と示した部分をクリックした時は掲示板システム全体に含まれる投稿を対象とし、2 と示した部分をクリックした時は現在表示されている掲示板カテゴリに含まれる投稿を対象とし、3 と示した部分をクリックした時は現在表示されているスレッドに含まれる投稿を対象とする。

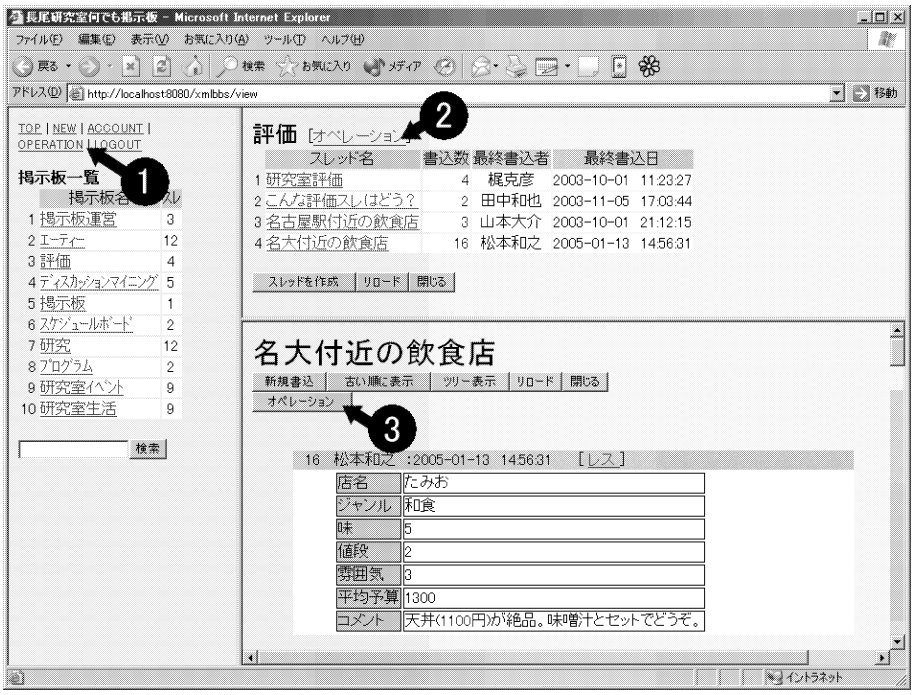

図 4.2: 検索対象の選択 (第一段階)

第二段階は、第一段階で選択した対象の中に含まれる投稿すべてか、特定のクラスに属する投稿のみかの選択を行う。第一段階の選択を行うと、図 4.3 で示すようなフォームが表示されるので、どのクラスに属する投稿を検索対象としたいかを選択する。このフォームで選択肢として現れるクラスは、第一段階で選択した対象に含まれるクラスのみである。例えば、「評価. 飲食店」クラスに属する投稿と「評価. 駐車場」クラスに属する投稿しか含まれていないスレッドを第一段階として選択した場合は、第二段階では、「すべての」クラス・「評価. 飲食店」クラス・「評価. 駐車場」クラスの 3 つしか選択肢として現れないようになっている。

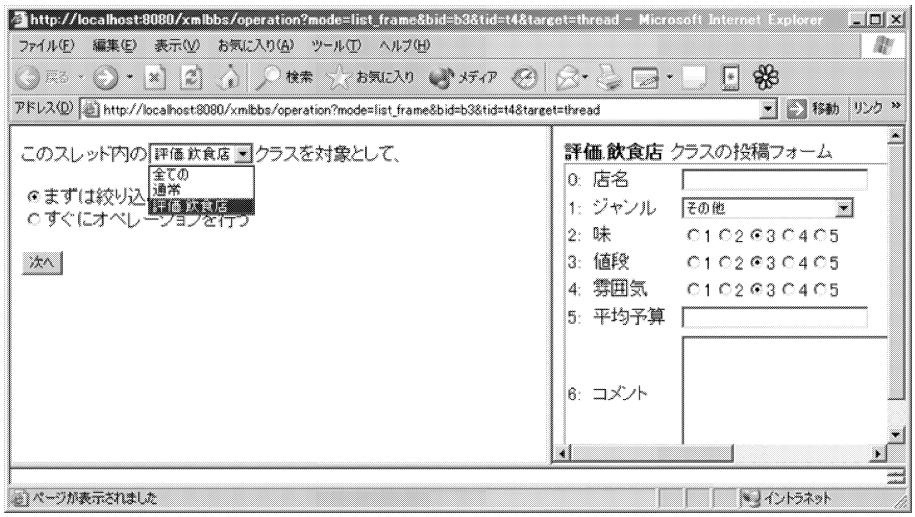

図 4.3: 検索対象の選択 (第二段階)

4.3 絞り込み条件の設定

次に、前節で選択された検索対象に対して、絞り込み条件を設定することでユーザの求める条件に合った投稿に絞り込むことができる。

前節の第2段階の検索対象として任意のクラスを指定した場合は、そのクラスに含まれるプロパティの値を用いた条件設定が可能となる。例えば「評価.飲食店」クラスに含まれるプロパティである「平均予算」の値が 800 以下である投稿、といった条件を設定できる。また AND 検索も可能で、「平均予算が 800 (円) 以下の中華料理の店」といった検索も可能である。

検索対象として「すべての」クラスを選択した場合は、string の属性を持った「タイトル」と「本文」の各プロパティの値を用いて、含まれる文字列のみの情報を用いて絞り込みを行うことができる。

条件を設定するとき、対象となるプロパティの型によって指定できる条件の種類が変化する。例えば、文字列型の値を持つプロパティに関する条件は、パラメータとして与える文字列と「完全一致する」「含む」などの条件を指定できるが、数値型の値を持つプロパティでは、パラメータとして与えた値との大小比較などによって絞り込み条件を指定する。

条件を設定する画面では、例えば「評価.飲食店」クラスを検索対象としたとす

ると、図 4.4 のように、そのクラスに属するプロパティのみが対象として選択できる。条件設定の対象とするプロパティとして「平均予算」という整数型のプロパティを選択しているとき、選択できる条件は図 4.5 のように、整数型に合った条件のみ選択できる。AND 検索をする場合は「条件の追加」ボタンをクリックすることによって、図 4.6 のように条件の追加を行うことができる。1 つめの条件と 2 つめの条件で同じプロパティに関する条件を設定することができるので、「平均予算が 1500 円以上かつ、3000 円以下」といった条件も設定可能である。

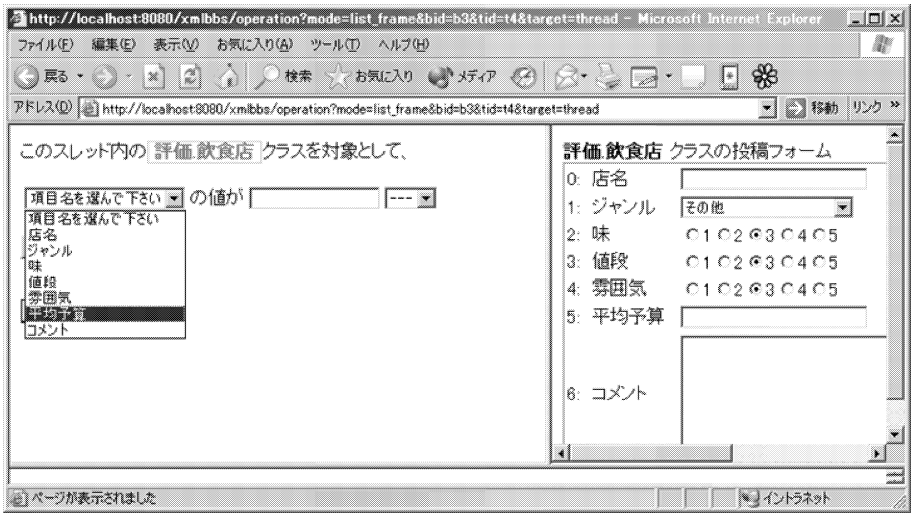

図 4.4: 選択できるプロパティ

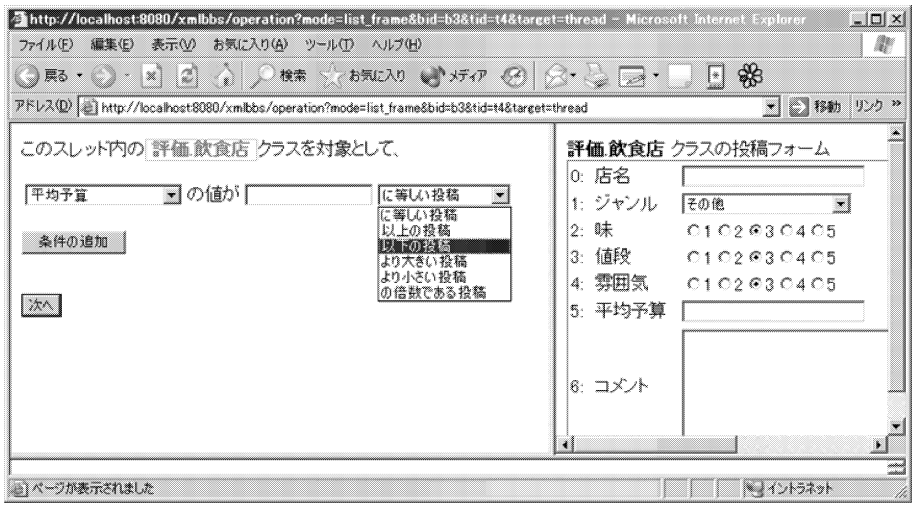

図 4.5: 選択できる条件

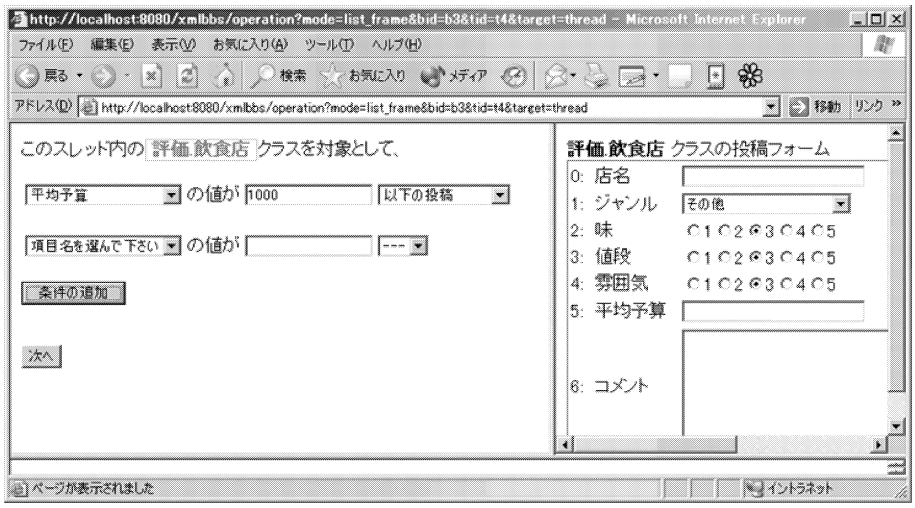

図 4.6: 条件の追加

4.4 絞り込み検索結果の表示

絞り込み条件の設定が完了したら、絞り込みの結果を表示できる。このとき、絞り込み結果の表示順序を設定することができる。特に表示順序にこだわらずに表

示する他に、数値の属性を持つプロパティの値によるソートを行うことができる。例えば、「ジャンルが中華の店」という条件で絞り検索を行ったときに、「平均予算の値が小さい順」といった表示順序の設定が可能である。このようにして、絞り込み検索の結果を表示した画面例を図 4.7 に示す。

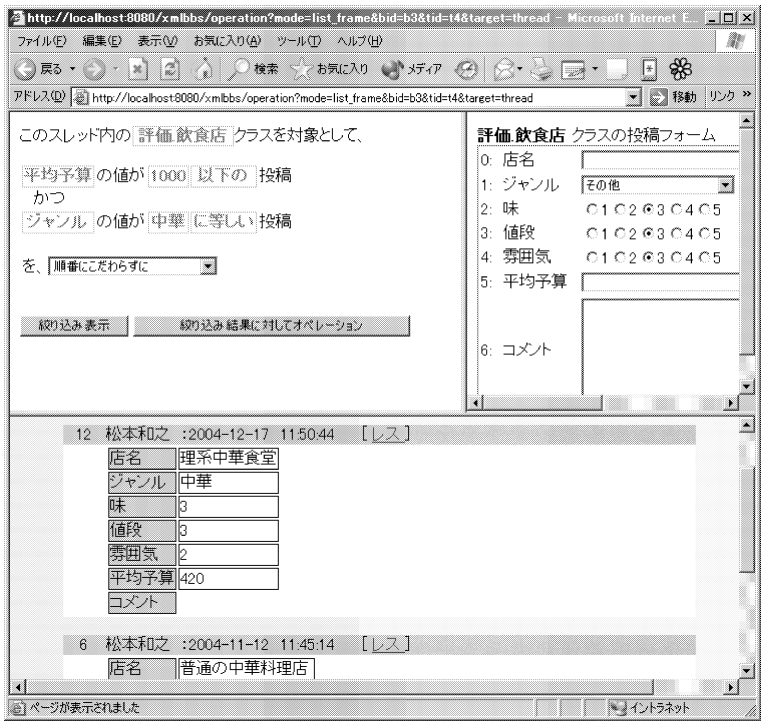

図 4.7: 絞り込み検索の結果

これまでに説明したように、投稿データに付与された構造化アノテーションにおけるプロパティを用いることによって、高度な絞り込み検索が可能となった。しかし、これでユーザのすべての要求に応えられるわけではない。そこで次節では、より高度な絞り込み条件をユーザ定義する手法について述べる。

4.5　絞り込みルーチンのユーザ定義

前節までに投稿データのプロパティに関する条件を設定する手法を説明したが、各プロパティの組み合わせで設定できる条件には限界がある。そこで、より柔軟な絞り込み条件を設定できるように、絞り込みルーチンをユーザが定義すること

も可能にした。

まず、ユーザがより柔軟に、自由な絞り込みルーチンを記述するためには、プログラミング言語で記述するのが適切であると考えられる。次に、どのような言語を用いるのが妥当であるかという問題であるが、独自に言語を設計するのは、必要な機能のみを自由に設定できる反面、その言語をユーザが習得しなければならないため、設計のコストに見合う利益は得られないと考えられるので、既存の言語を利用することにした。では、どの言語が良いかを考えたとき、次に挙げる条件を満たしていることが望ましいと考えられる。

1. 幅広く知られた言語であること

2. コンパイルの必要ないインタプリタ型スクリプト言語であること

3. サーバサイドで実行される言語であること

1は、多くのユーザが記述できるために重要であると考えられる。2は、コンパイルせずに済む点と、プログラムの修正と実行がスムーズに行えるという点において優れていると考えられる。3は、実行結果をサーバ側で管理したり蓄積したりするために必要であると考えられる。以上のような条件を満たす言語として、本研究ではPerl [17] を採用した。内部的には、サーブレットがPerlを呼び出しているに過ぎないので、Perl以外の言語に差し替えることも可能である。

以上のような理由から、本システムにおいて絞り込みルーチンをユーザ定義する際には、絞り込みルーチンをPerlスクリプトで記述する。その際、スクリプトはサブルーチンの形式で記述し、戻り値として投稿の通し番号のリストを返すようにしておくことにより実現される。またこれは、前節で述べた絞り込み検索結果の表示順序に関しても独自に決められる。すなわち、サブルーチンの戻り値であるリストの順序が、検索結果の表示順序となる。単純な絞り込みルーチンの例を以下に示す。このルーチンは、「評価.飲食店」クラスを対象とし、味の評価値・値段の評価値・雰囲気の評価値の合計が10以上の投稿で絞り込むルーチンである。なお、スクリプト中では表4.1に示す変数を自由に利用することができる。

```
sub getGoodShop {

  my($sum)=0;
  my($i)=0;
  my(@ret)=();

  for($i=0;$i<$_num;$i++){
    $sum = $_data[$i][2] + $_data[$i][3] + $_data[$i][4];
    # 合計が10以上なら戻り値リストに投稿番号を追加
    if($sum>=10){
      push(@ret,$i);
    }
  }
  return @ret;
}
```

ここで、表中の変数 i と j について説明する。4.2 節で、絞り込み検索の対象となる投稿集合の決定について述べたが、この投稿集合に含まれる投稿は、0,1,2,…の順に通し番号がついている。この通し番号が表中の変数 i に相当する。また、1つの投稿に焦点を当てると、これには 1 つ以上のプロパティが含まれる。このプロパティにも通し番号 0,1,2,…が付いている。このプロパティの通し番号に相当する変数が j であり、変数$_data を用いることによってプロパティの持つ値を参照できる。

以上のようにしてユーザが任意の絞り込み条件を設定することができ、また記述したスクリプトを他のユーザが利用したり拡張したりすることも可能である。

4.6　複数クラスにまたがった絞り込み検索

さらに、プロパティのオーバーライドの概念によって、先祖・子孫関係にある複数クラスにまたがった絞り込み検索もできる。例えば、「営業時間」というプロパ

表 4.1: スクリプト中で利用可能な変数

変数名	データ型	説明
$_num	整数	対象となる投稿の数
$_name[i]	文字列	i 番目の投稿の投稿者名
$_date[i]	文字列	i 番目の投稿の投稿日
$_time[i]	文字列	i 番目の投稿の投稿時間
$_classname[i]	文字列	i 番目の投稿の投稿クラス名
$_data[i][j]	文字列	i 番目の投稿の、j 番目のプロパティ値

ティを持つ「店舗情報」クラスのサブクラスとして「飲食店情報」「駐車場情報」「アミューズメント施設情報」があるとする。すべてのサブクラスは、その親クラスの属性「営業時間」をオーバーライドしているため、店舗情報クラスを対象として「8:00 以降まで営業している店舗」という条件で検索することによって、「飲食店情報」「駐車場情報」「アミューズメント施設情報」といった異なるクラスであるにも関わらず、それらの中から条件に合った店舗情報を検索することができる。

第5章　投稿データに対するオペレーション

5.1　オペレーションとその手順

投稿データは、蓄積しただけでは役に立たず、効率よく利用する手法について考える必要がある。本研究では、蓄積された投稿データを処理し何らかの結果を出す操作をオペレーションと呼び、単純なアンケートの集計やマーケティングなど、様々な知識発見のために利用できる。

オペレーションには自由度の高さの観点から3通りの方法が可能である。どの方法においても、オペレーションの手順は概ね以下の通りである。まずオペレーションを行う対象の選択を行い、次にオペレーションの選択または構築を行う。その後、オペレーションの結果を表示することができる。オペレーションを作成した場合は、必要に応じて登録できるほか、次章で説明する、オペレーション結果の投稿を行うことも可能である。

5.2　オペレーション対象の選択

オペレーションを行うために、まずオペレーション対象となる投稿の集合を決定する。オペレーション対象としては、(a)4.2節で説明した絞り込み検索の対象と同じもの、あるいは、(b) 絞り込み検索の結果、のいずれかを選択できる。

(a) では、掲示板全体・任意の掲示板カテゴリ・任意のスレッドを選択し、さらにその中のすべての投稿か、任意のクラスに属する投稿のみかを選択した結果を投稿集合にできる。これは、4.2節で説明した第一段階の対象を選択したときに現れる図5.1のようなフォーム（この図はスレッドを対象として選択している）で、「すぐにオペレーションを行う」という項目にチェックした状態で「次へ」をクリックすれば良い。クラスを指定したい場合は、4.2節と同様に、リストボックスから任意のクラスを選択した状態で「次へ」をクリックすれば良い。

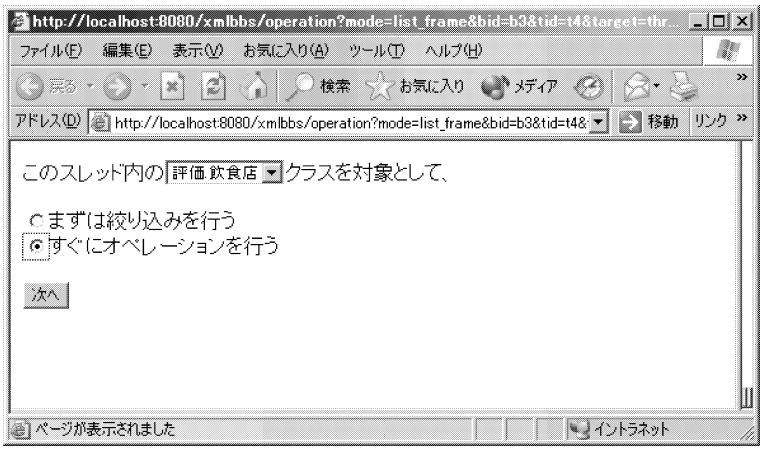

図 5.1: 絞り込みを行うかどうかの選択画面

一方 (b) では、絞り込み検索の実行結果として得られた投稿集合を対象とできる。これは、4.2 節と 4.3 節で説明したような GUI 上で設定した条件に対する絞り込み検索結果にも、またユーザ定義の絞り込みルーチンによって出力された絞り込み検索結果にも適用できる。具体的には、絞り込み検索結果の表示画面の下部フレームにある「この結果に対してオペレーション」というボタンをクリックすることにより、(b) を対象とできる。

5.3　3通りのオペレーション

オペレーション対象を決定したら、次にその対象に対してどのようなオペレーションを行うのかを決定する。オペレーションは 3 通りの方法で行うことができ、それぞれ自由度の高さと簡単さが異なり（図 5.2)、幅広いユーザ層に対応できると考えている。

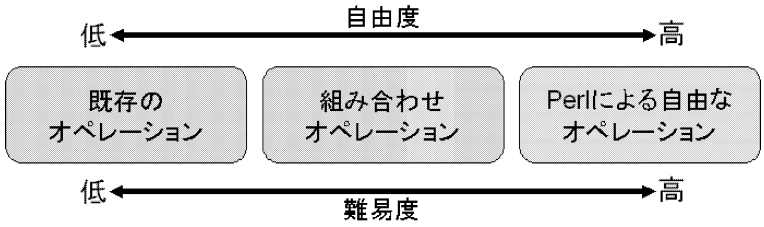

図 5.2: 3通りのオペレーション

5.3.1 方法1：他のユーザが作成したオペレーションの利用

もっとも簡単に実行できるオペレーションは、他のユーザが作成したオペレーション一覧の中から実行したいオペレーションを選択し、実行する方法である。図5.3に示すように、現在選択されているオペレーション対象に対して実行可能なオペレーションのリストが表示されるので、この中から実行したいオペレーションの左にあるラジオボタンをチェックした後、「オペレーション実行」ボタンを押すことによりオペレーションが実行される。この方法は、他のユーザが作成したオペレーションをそのまま実行するだけなので、ユーザの求めるオペレーションが存在しなければ、実行できないという、自由度は低い方法である。

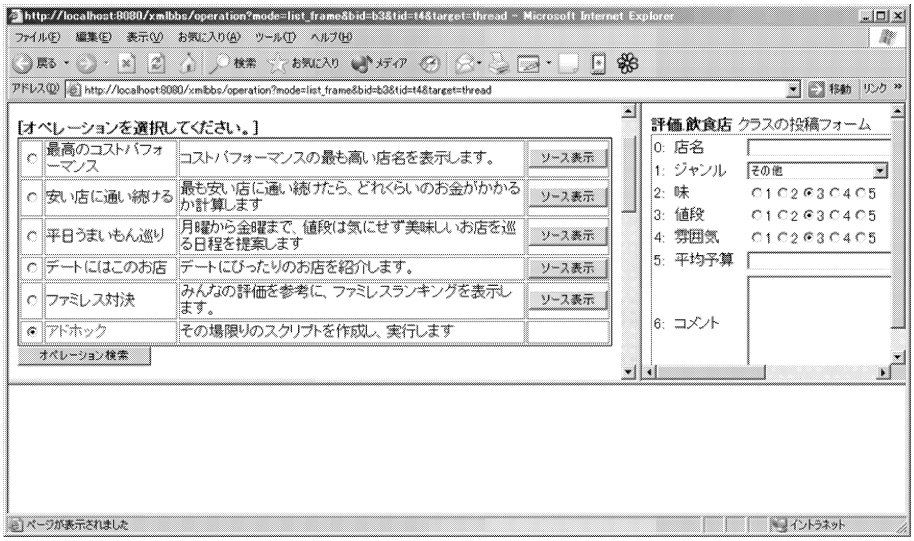

図 5.3: オペレーション一覧

図5.4は、このタイプのオペレーションを実行した画面の一例であり、ユーザ定義された「平日うまいもん巡り」というオペレーションが実行され、下部フレームにオペレーション結果が表示されている。

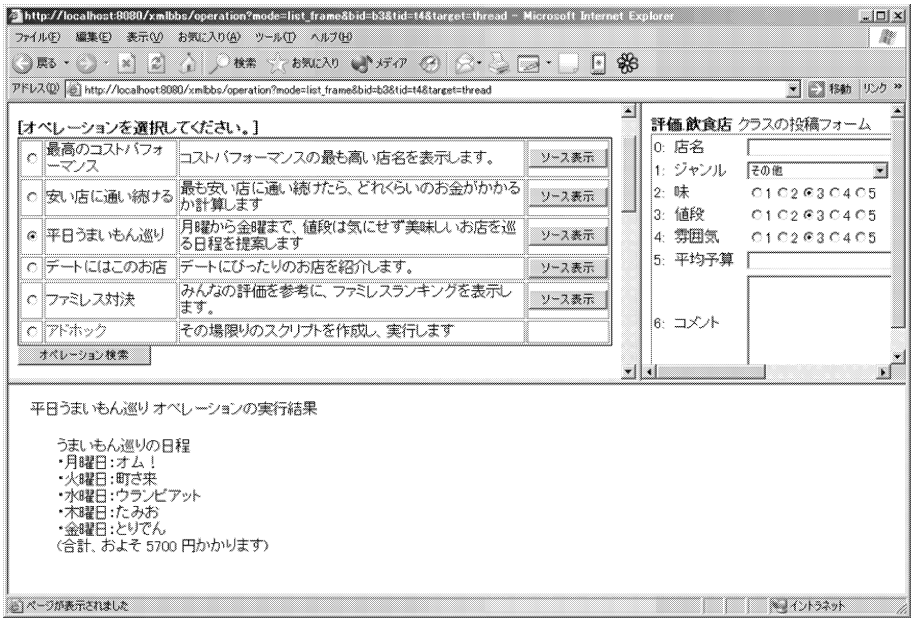

図 5.4: 「平日うまいもん巡り」オペレーションの実行結果

5.3.2 方法2：Perlスクリプトによるオペレーションの作成

もっとも自由度が高い方法は、Perlスクリプトを用いてユーザが自由に記述する方法である。Perlに関する知識があれば、各投稿の投稿内容・投稿クラス・投稿日時・投稿者などの情報を自由に用いてユーザの求める処理を記述することができる。

スクリプトは、図5.5に示すテキストエリア内に記述する。またスクリプト中では、表5.1に示す変数を利用して自由にコードを記述することができる。スクリプトにおける標準出力が、そのままオペレーションの実行結果の出力となるので、print文などによって容易に記述することができる。

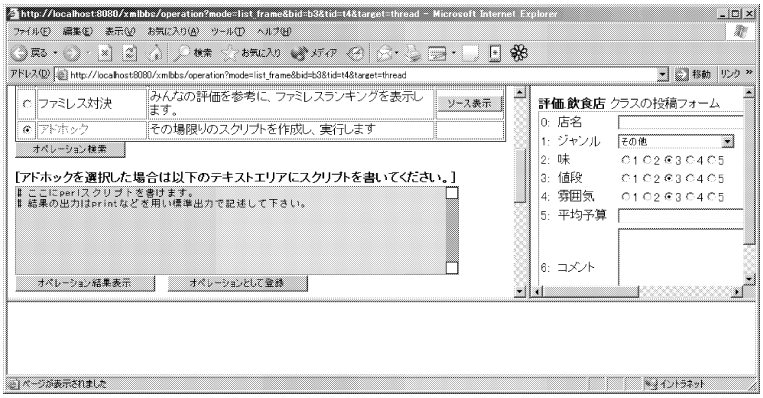

図 5.5: オペレーションを入力するテキストエリア

表 5.1: スクリプト中で利用可能な変数

変数名	データ型	説明
$_num	整数	対象となる投稿の数
$_name[i]	文字列	i 番目の投稿の投稿者名
$_date[i]	文字列	i 番目の投稿の投稿日
$_time[i]	文字列	i 番目の投稿の投稿時間
$_classname[i]	文字列	i 番目の投稿の投稿クラス名
$_data[i][j]	文字列	i 番目の投稿の、j 番目のプロパティ値

スクリプトを記述する際に、投稿クラスの構造情報が分かっていると大変便利であるので、右上フレームに、現在対象となっている投稿クラスの投稿フォームと、投稿内容を参照するためのプロパティの通し番号を表示するようにした。

以下に、「評価.飲食店」クラスを対象とした「安い店に通い続ける」というオペレーションのソースコードを示す。このオペレーションは、「投稿データの中から最も平均予算の安い店を探し出し、その店に毎日通った場合およそどれくらいの金額が必要になるか」を計算し、出力するオペレーションである。

```
#平均予算が最低の店を探す
$best_price = $_data[0][5];
$best_shop = $_data[0][0];

for($i=0;$i<$_num;$i++){
 $price = $_data[$i][5];
 if($price<$best_price){
  $best_price = $price;
  $best_shop = $_data[$i][0];
 }
}

#その店に通い続けたときにいくらかかるか計算
$cost_day = $best_price*3;
$cost_month = $best_price*3*30;
$cost_year = $best_price*3*365;

#結果の出力
print "一日三食、$best_shop に通い続けると、およそ\n";
print "１日で$cost_day 円、\n";
print "１ヵ月で$cost_month 円、\n";
print "１年で$cost_year 円かかります。";
```

このように、その場で記述してその場で実行するスクリプトを、アドホック・オペレーションを呼んでいる。実行する時はオペレーションの一覧の中から「アドホック・オペレーション」を選択した状態で、スクリプトを記述するテキストエリアにソースコードを記述し、「オペレーション実行」ボタンをクリックすることにより、下部フレームにオペレーション結果が表示されるので、出力を見ながらオペレーションの構築やデバッグが容易にできる。上に示した「安い店に通い続ける」オペレーションをアドホック・オペレーションとして実行した時の画面イ

メージを図5.6に示す。

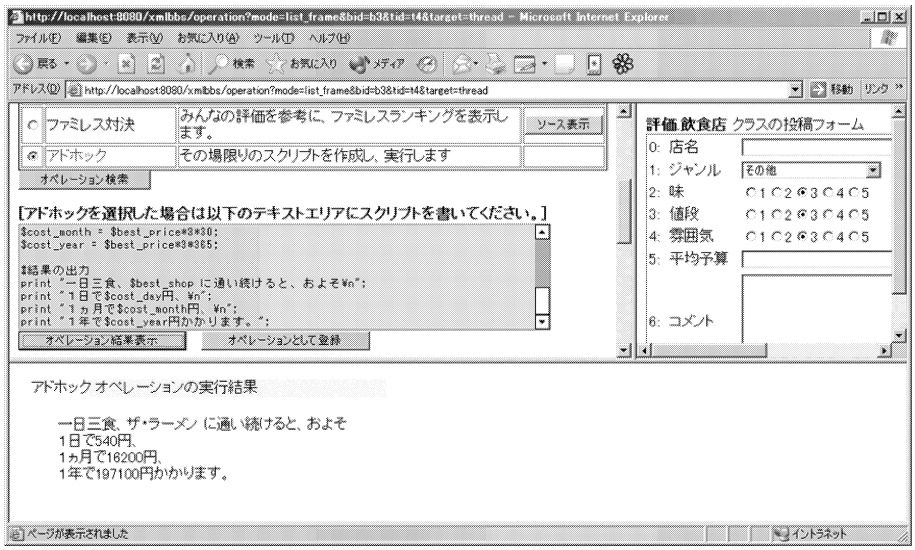

図 5.6: 「安い店に通い続ける」オペレーションの実行結果

作成したオペレーションを登録することによって、4.3.1節で説明した方法によって任意のユーザがそのオペレーションを実行できるようになる。また、他のユーザがそのオペレーションのソースコードを改変したり、参考したりすることによって、新たなオペレーションの作成に役立つかもしれない。実際に、他のユーザが作成したオペレーションのソースコードを見るには、オペレーションの一覧が表示されている画面において、ソースコードを見たいオペレーションの右部にある「ソース表示」というボタンをクリックするだけで、別ウィンドウにそのソースコードが表示される。オペレーションを登録するには、図5.7で示すように、オペレーション名とそのオペレーションの説明、対象クラス、実際のソースコードの情報をサーブレットに送信すればよい。なおオペレーションの説明は、オペレーションをキーワード検索する際にも用いられる。

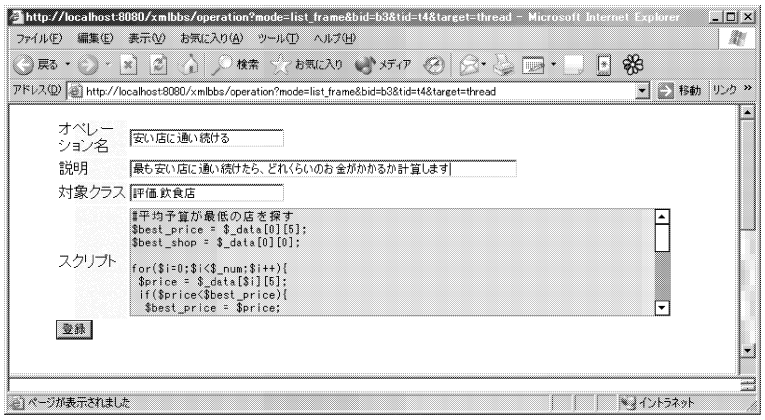

図 5.7: オペレーションの登録

5.3.3 サブルーチンの定義

Perl スクリプトを用いてサブルーチンを定義することにより、コードの再利用性を高めることも可能である。サブルーチンとして登録しておくことにより、すべてのユーザがオペレーションを記述するときに利用することができるようになる。

Perl におけるサブルーチンは、引数を取ったり戻り値の指定ができ、また本システムにおいても引数や戻り値に対応している。引数は Perl において一般的な形である $_[0],$_[1],… といった変数をサブルーチン内に記述することによって、自動的に引数の存在や引数の数を認識する。また戻り値も return 文を用いるだけで設定できる。

サブルーチンは図 5.8 に示すフォームに記述し、使用できる変数はオペレーションのスクリプトを記述する時と同様のもの（表 5.1）となっている。サブルーチンを記述しながら、適宜「戻り値表示」ボタンをクリックすることによって実行結果 (サブルーチンの戻り値) を確認することができるので、スムーズにサブルーチンを構築できる。

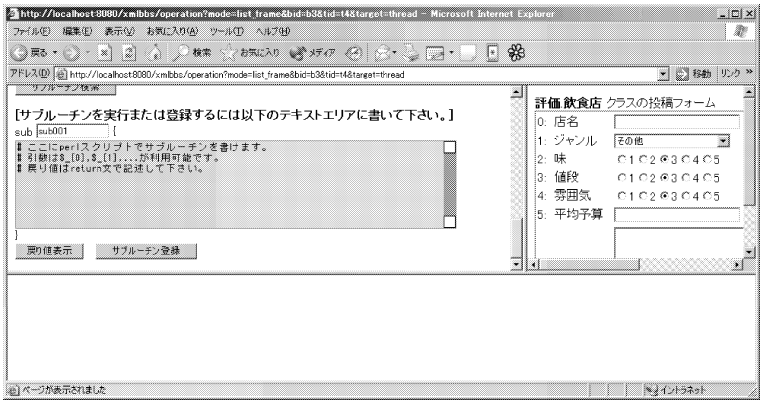

図 5.8: サブルーチンを入力するテキストエリア

現在対象として選択されているクラスに適用できるサブルーチンは、図5.9のようにサブルーチン一覧として表示される。最も左の列はサブルーチン名を表している。左から2番目の列はそのサブルーチンの説明であり、説明中にあるテキストボックスは引数があることを示している。左から3番目の列には、そのサブルーチンを実際にソースコード中で利用するにはどのように記述すれば良いのかを表しており、サブルーチンに引数が存在する場合は、サブルーチンの説明中にあるテキストボックスに引数を入力・変更すると、随時サブルーチンの引数表示が変更される（図5.10）。

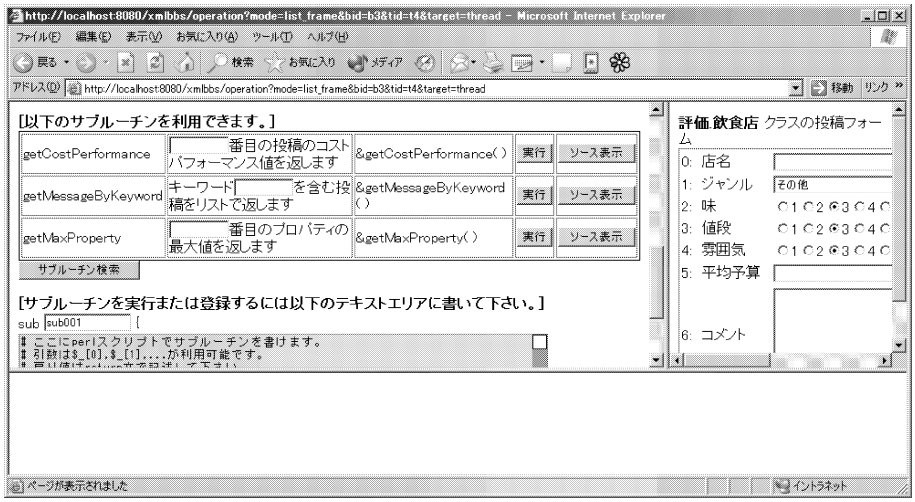

図 5.9: サブルーチン一覧

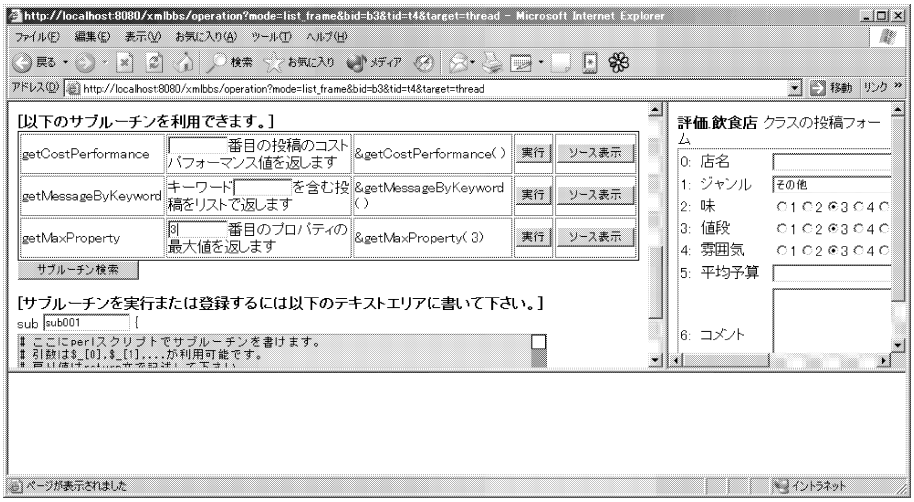

図 5.10: 動的に変わるサブルーチンの引数表示

登録されているサブルーチンを実行し、戻り値を調べたい場合は、「実行」ボタンをクリックする。このとき、引数情報も利用されるので引数の必要なサブルーチンや、引数を設定可能なサブルーチンでは説明中のテキストボックス中に引数を記述した状態で「実行」ボタンをクリックする。

「ソース表示」ボタンは、アドホック・オペレーションの構築時と同様に、そのサブルーチンのソースコードを表示させることができる。

サブルーチンを登録するには、サブルーチンを記述するフォームに、サブルーチン名とサブルーチンの内容を記述した状態で、「サブルーチン登録」ボタンをクリックする。すると図 5.11 に示すフォームが現れるので、サブルーチン名・サブルーチンの説明・引数の数などを設定したうえで「登録」ボタンをクリックすれば良い。このサブルーチンの説明中に、$_[0],$_[1],…といった Perl において引数を表す変数を含めることにより、先に説明したように、サブルーチンの説明文中にテキストエリアとして引数を記述できるようになる。またサブルーチンをキーワード検索するときにも、サブルーチンの説明が用いられる。

ただし、同一のサブルーチン名が存在する場合は登録できない。

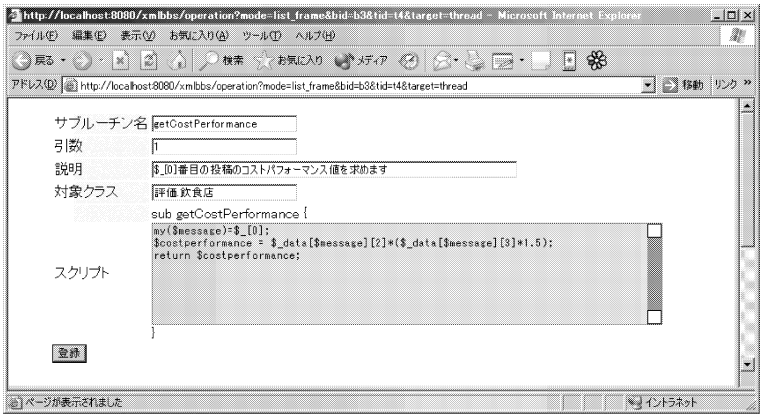

図 5.11: サブルーチン登録フォーム

5.3.4 方法3：APIやサブルーチンの組み合わせによるオペレーション

これまで説明してきた2つの方法の中間的な位置づけになる方法は、既存の標準APIやサブルーチンをGUI上で組み合わせて構成する方法である。本システムでは表5.2に示すような標準APIが実装されており、数値プロパティであれば平均・分散・標準偏差などを求めるAPIや、文字列であれば形態素解析結果を返すものがある。これらとサブルーチンを組み合わせることによって、GUI上でオペレーションを構成できる。

図5.12に示すのがオペレーションを構築するためのGUIである。オペレーションは、プログラミング言語のように順に実行され、図中の(1)(2)(3)の順で実行される。例えば(1)の行では「表示する」が命令であり、「合計はおよそ」の部分がパラメータである。すなわちこの行は、「合計はおよそ」という文字列を表示する、という意味を持っている。右の「編集」ボタンをクリックすることにより、その内容が編集できるモードになる。図では(2)の行が編集モードになっており、命令の選択やパラメータの編集が行える。(2)の行のパラメータで「(入力一覧から選択して下さい)」と表示されている部分が現在編集を行っている部分で、選択されている部分に入るパラメータを、下部にある入力一覧から選択できる状態になっている。入力一覧には、サブルーチンの他に、本システムに実装されている標準APIや、他の行での計算結果、静的な文字列や数、投稿における各プロパティの

表 5.2: 実装されている API の一部

表記	説明
&getSum(n)	n 番目のプロパティ値 (数値) の合計値を返す
&getAverage(n)	n 番目のプロパティ値 (数値) の平均値を返す
&getVariance(n)	n 番目のプロパティ値 (数値) の分散を返す
&getMessageWhenMax(n)	n 番目のプロパティ値 (数値) が最大となるときの投稿番号を返す
&getMorphologicalAnalysis(s)	文字列 s の形態素解析結果を返す
&sortBy(n)	n 番目のプロパティ値を用いて投稿をソートした結果を返す

値などがあり、それらの中から選択することでパラメータとして入力できる。このようにして、プログラミング言語に関して全く知識がなくても、オペレーションを構築できるが、スクリプトを書いて構築するオペレーションのような自由度の高さはない。

図 5.12: オペレーション構築 GUI

第6章　オペレーション結果を用いた知識発見の支援

6.1　オペレーション結果の投稿

ユーザがオペレーションを行ったとき、その結果に意味があったとしても無かったとしても、結果を見てオペレーションを終了するのではなく、オペレーション結果＝知識を他のユーザと共有することは有意義であると考えられる。本システムでは、オペレーションの実行結果を直接投稿することが可能である。もちろん、実行結果をコピー・ペーストすることでも投稿できるしれないが、その結果を再利用しようと思ったときに、構造化アノテーションされていた方が都合がよい。

そこで本掲示板システムでは、投稿結果を直接構造化アノテーション付きで容易に投稿できるようにした。オペレーションのタイプや種類に関係なく、オペレーションの実行結果の画面で「この結果を投稿する」ボタンをクリックすることにより、オペレーション結果の投稿フォームが現れる（図6.1）。次に、このフォームのコメント欄に、オペレーションの結果を見て思ったことや考えたこと、発見したことなどを、任意で入力する。最後に「送信」ボタンをクリックするだけで、オペレーション結果の投稿を行うことができる。

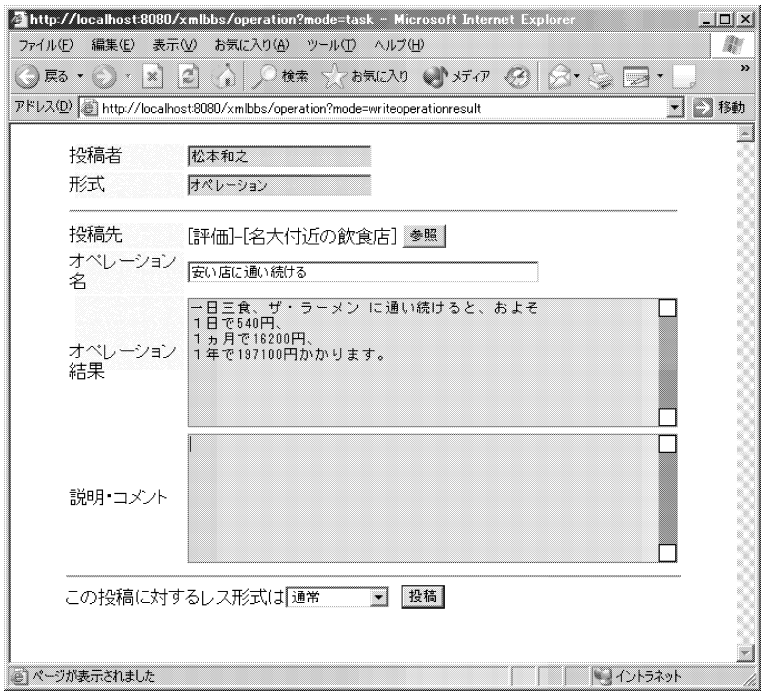

図 6.1: オペレーション結果の投稿フォーム

6.2 オペレーション自身の投稿

ユーザがオペレーションを行い、興味深い結果が得られたとする。しかし、1 週間後に同じオペレーションを実行した時にも、まったく同じ結果が得られるだろうか。答えはおそらく No である。なぜなら、電子掲示板において投稿データは増え続けるものだからである。オペレーションを実行するときに使用する投稿データが変われば、当然オペレーション結果も変わるであろう。このように、オペレーション結果は時間と共に変動するものである。あるスレッドにおいて、意味のあるオペレーションが任意のユーザによって任意のタイミングで実行できることは、知識発見という観点から大きな意味があると考えられる。

そこで本システムでは、オペレーション自身を投稿として投稿できる機能（これを「オペレーション投稿」と呼ぶ）を実装した。5.3.2 節で述べたオペレーションのユーザ定義は、オペレーションを登録することによって、対象クラスの投稿に対してオペレーションを行う他のユーザに対してもそのオペレーションが一覧に表示され、利用できる。一方オペレーション投稿は、スレッドに対してオペレー

ションを投稿するだけなので、そのスレッドの範囲内でしかあまり意味を持たないオペレーションを、ローカルで行うのに適している。オペレーション投稿を行うと、それを閲覧したときに、オペレーションの説明とともに「オペレーション実行」ボタンと「ソース表示」ボタンが表示される（図6.2)。ユーザが「オペレーション実行」ボタンをクリックすると、その時点での投稿データを用いて、投稿されたオペレーションを実行し、結果が表示される。例えば、賛成か反対かを投稿するスレッドがあり、賛成の数と反対の数の集計を行うオペレーションが投稿されているとする。このとき、オペレーション投稿にある「オペレーション実行」をクリックすることによって、クリックした時点での賛成の数と反対の数の集計結果が表示されるので、その時点での状況を容易に把握できる。

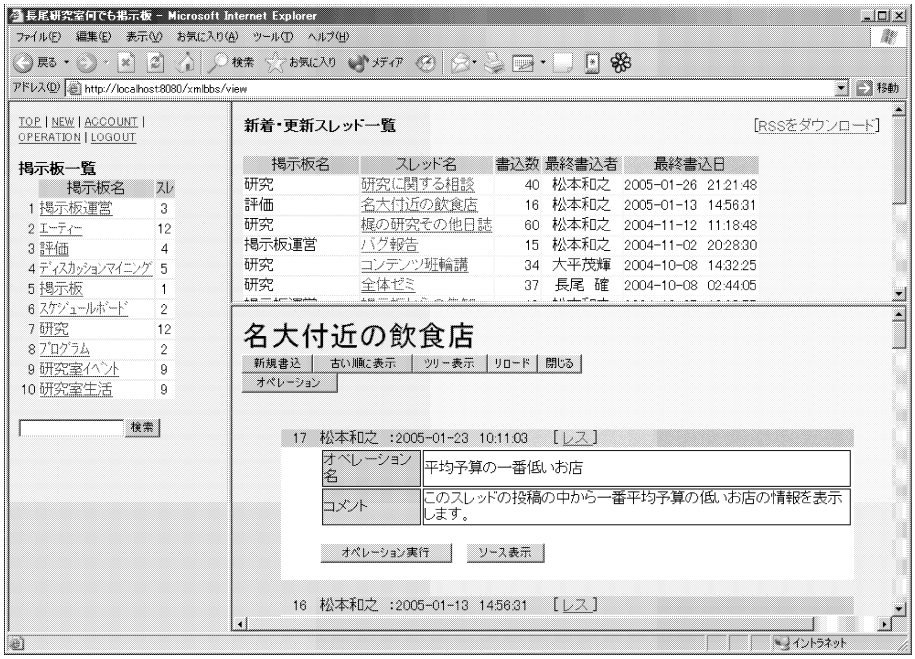

図 6.2: オペレーション投稿の表示

6.3 オペレーションデーモン

前節で説明したように、ユーザが任意のタイミングでオペレーションを実行するのではなく、定期的に、途中経過などの情報に関して、システムの方から自動的にオペレーション結果を投稿としてプッシュするのが、「オペレーションデーモ

ン」という機能である。前節の賛成・反対の例で言えば、賛成の数と反対の数を、ユーザが知りたいと思った任意のタイミングでオペレーションを実行するのではなく、例えば毎日 0:00 になると、システムが自動的に投稿という他のユーザが見られる形で提示する。投稿という形で提示すれば、オペレーションを実行しないユーザでもそのスレッドを閲覧すれば自然に目に入るため、ユーザに知っておいて欲しい情報などを、人間の手を介すことなく提示できることは有意義であると考えられる。

オペレーションデーモンは、投稿を行うスレッドと、実行するオペレーション、実行するタイミングをそれぞれ指定することによって設定される。スレッドに含まれる投稿一覧を表示した画面で「オペレーション」ボタンをクリックし、オペレーションを実行する画面まで進んだら、実行したいオペレーションを一覧の中から選択した後、「オペレーションデーモン」ボタンをクリックすることにより、図 6.3 に示すオペレーションデーモンの設定を行うフォームが出現する。この時点で、投稿を行うスレッドと実行するオペレーションは選択されているので、フォーム上で実行するタイミングを決定すればオペレーションデーモンの設定が完了する。

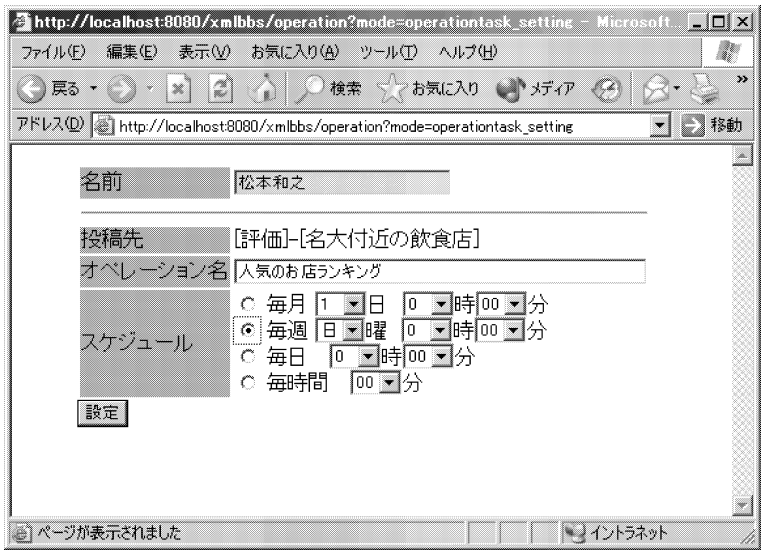

図 6.3: オペレーションデーモンの設定

第7章　関連研究

7.1　TelMeA

TelMeA [25] は、コミュニティ上で行われるコミュニケーションをより容易に、より豊かにする、avatar-like エージェントを用いたコミュニティシステムである。TelMeA は、電子掲示板やメーリングリストのような文書主体のコミュニティシステムではなく、ユーザの存在や主体性を表すアバターを用い、そのアバターに発話させたり、身振り・表情・指さしといった身体的な動作によってコミュニケーションを行う形式の会話インタフェースを持ち、図 7.1 のような手順で台本 (投稿) を作成する。アバターが身振りなどの設定に従ってアニメーションしながら発言し、それに返信したユーザの投稿にもアバターが登場し、対話しているような形式でスレッドを閲覧できる。

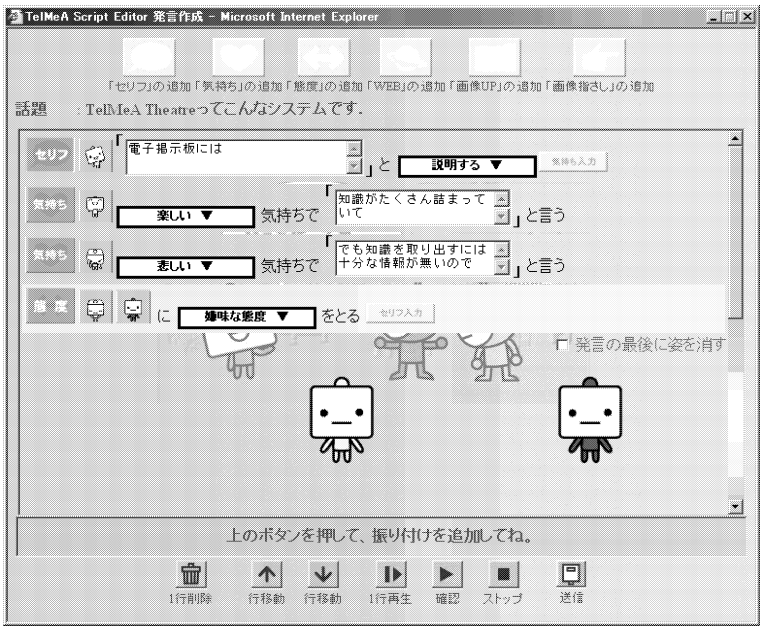

図 7.1: TelMeA における投稿の作成

TelMeA は、システム的には電子掲示板に近い部分が多く、アバターに発言させるか文章として書くかの違いはあるものの、ユーザによってスレッドが作成され、スレッドに投稿でき、投稿に対して返信できる。また、アバターの発言 (本文) に対して身振り・表情・指さしといった付加的な情報 (アノテーション) を付与するという点で、本研究と類似していると考えられる。しかし、TelMeA ではそういった情報の使用に関しては閲覧時のアニメーション程度にしか用いられておらず、蓄積された投稿データをどのように利用するかという部分については触れられていない。

7.2　機械学習による電子掲示板からの評判情報抽出

東京大学の藤村ら [22] は、インターネット上に溢れる評判情報を電子掲示板から抽出するためのルールを機械学習によって発見し、商品などの評判の善し悪しや評判を表す形容詞などを抽出する研究を行っている。具体的には、電子掲示板におけるコメント一文一文を、評判 (良)・評判 (悪)・その他の 3 つに分類し、C4.5 を用いることによって判別ルールを作成している。

本研究では投稿に対して人手で構造化アノテーションを付与し、その情報を用いることによって評判情報の抽出などのオペレーションを行うため、現在運用されている掲示板の投稿データを用いてオペレーションを行うことはできない。しかし、ここで挙げている研究においては、全て機械学習によって情報を抽出するため、現在運用されている様々な電子掲示板に対して適用できるが、正解率を常に考慮する必要があるという問題、また、評判以外の情報を抽出しようと思ったときに再びルールを機械学習によって構築する必要があるという問題を含んでいる。

第8章　今後の課題

8.1　オペレーション結果表示インタフェースの拡張

現在、投稿データを用いてオペレーションを行った結果は、Perl における print 文で出力する仕様のため、テキストによる出力以外は困難な状況である。例えばアンケートの集計結果は、テキストで割合を表示するよりも、グラフなどを用いたグラフィカルなインタフェースによって出力する方が視認性がよく、理解がより容易であろう。

グラフィックを出力する方法として SVG (Scalable Vector Graphics) [19] が適していると考えられる。SVG は、XML 形式で記述されるベクタグラフィックスであり、2001 年 9 月に W3C 勧告となった。オペレーションによるグラフィック出力は、グラフや図形の組み合わせなど、多くはベクタグラフィックスに適した形式であろう。また、SVG は XML で記述されるため、データベースに保存されているデータのほとんどが XML 形式である本システムとの親和性も高いと考えられる。

実装について考えると、SVG をオペレーションで初めから構築するのは容易なことではないので、円グラフを作る API や、折れ線グラフを作る API、多角形を描画する API、文字を入れる API など、SVG 形式でグラフィックスを容易に描画できる様々なインタフェースを用意すると良いであろう。そうすれば、例えば、

```
&svg_makeCircleGraph('賛成',$agreeDegree,'反対',$disagreeDegree);
```

と記述するだけで、賛成と反対の割合を示す円グラフが描画できるようになる。

以上のように、出力としてグラフィックスも利用可能となれば、オペレーション結果の表現方法により広がりができるであろう。さらに、将来的には、出力はテ

キストやグラフィックだけでなく、動画や音声といったマルチメディアコンテンツとすることも可能であろう。

8.2　スクリプトのエラー処理とセキュリティ

本システムでは、絞り込み検索ルーチンやオペレーションをユーザ定義する際に、Perl スクリプトを用いることができるが、スクリプト中でエラーが発生した場合の対処は十分に考えられていない。現在、エラーが発生すると絞り込みの結果は空になり、またオペレーションの結果は何も表示されないので、ユーザは十分にスクリプトのデバッグを行えない。これを改善するためには、Perl を実行した時に出力されるエラーメッセージを受け取り、ユーザに提示する必要がある。また、エラーの発生した行番号を取得して、スクリプトを入力するテキストエリアのどの行であるかを示すことによって、ユーザによるデバッグ作業の手助けとなるであろう。

また、Perl スクリプトが利用可能なことによって、投稿データを用いた非常に自由度の高い様々なオペレーションが可能となったが、その一方でセキュリティの問題が避けられない。例えば、Perl を用いてローカルディスクにアクセスすることも可能であるし、他サーバとのソケット通信を行うことも可能となる。こういったセキュリティ問題を解決するためには、スクリプト中の特定のキーワードに着目して、実行できないような対策が必要となる。例えば open 関数は、ディスクアクセスする際にファイルオープンする関数であり、これが含まれているスクリプトは実行しないようにする必要がある。これと同時に、フォルダに対するアクセス権の設定もしておいた方が良いであろう。他にも、ソケット通信関係の関数として、socket・bind・listen などといった関数も open 関数と同様に扱う必要がある。

バグを含んだスクリプトを実行することによって、無限ループに陥ってしまうかもしれない。それによってサーバがダウンしてしまうかもしれない。これは、多くの共用サーバのように「ユーザの責任で」という方針で良いと考えている。掲示板システムは誰でも利用できるわけではなくログインが必要であるため、どのユーザによって、そういったバグを含んだスクリプトが実行されたのかという情報が取得できるからだ。また、サーバのダウンを未然に防ぐために、常にプロセスの CPU 使用率やその時間を監視し、長時間 CPU を占有し続けるようなスクリプトが実行されていたら、プロセスを強制的に終了する方法も考えられる。

8.3 複数掲示板でのデータ共有

本掲示板システムが複数存在したとき、投稿クラス・投稿データ・オペレーションやサブルーチンに関する情報を共有できることは非常に有益であると考えられる。投稿クラスを共有できれば、同様の目的をもったユーザ定義の投稿クラスが他の掲示板システムに存在したとき、そのクラスをそのまま引用できる。このことによる利益は非常に大きいと考える。

まず、オペレーションやサブルーチンは適用可能な投稿クラスが決まっているため、引用した投稿クラスに対して適用可能なオペレーションやサブルーチンも同様に全て引用できるので、引用した時点から豊富なオペレーションやサブルーチンを利用できる可能性がある。さらに引用元の電子掲示板システムでは、引用した投稿クラスの投稿データが蓄積されている可能性が高いので、そのデータを含めたオペレーションやサブルーチンを行うことが可能となる。このように様々なデータを共有することにより、異なるコミュニティ間で知識を共有することができ、それが新たな知識発見に繋がるであろう。

第9章　おわりに

従来の電子掲示板システムでは、投稿がプレーンテキストであったため、単純なテキストマッチングによる検索程度しかできず、投稿データの再利用性は極めて低かった。本論文では、構造化アノテーションを投稿に対して付与することによって、検索を始めとする様々な高度な処理を効率的に行える、投稿データの再利用性が高い電子掲示板である、アノテーション掲示板システムについて述べた。

任意のユーザは、投稿クラスを定義することによって投稿に対する構造を自由に定義することができ、その定義から自動生成される投稿フォームに入力するだけで、構造化アノテーション付きの投稿を行うことができた。蓄積された投稿データは、絞り込み検索やオペレーション等によって各ユーザの目的に応じた方法で利用可能にした。絞り込み検索は、検索対象を選択した後、システムで用意されるGUIまたはPerlを用いてユーザ定義したサブルーチンによって自由な条件の設定が可能である。また、オペレーションは、その自由度や難易度の異なる3通りの方法を用意することにより、幅広いユーザ層に対応した。さらに、オペレーション結果の直接投稿や、オペレーション投稿、オペレーションデーモンによって、知識発見の支援を行うことができる仕組みとなっている。

以上のように、本研究では、投稿構造を自由に設定できるので用途が限定されず、また投稿データの利用方法も限定されない仕組みを全て電子掲示板の枠組みの中で構築した。

本論文で提案したアノテーション掲示板システムが普及し、あらゆる種類の投稿が構造化されれば、提示版上のあらゆる情報が整理され、それらの情報を誰でも容易に高度に利用できる情報システムが実現されるであろう。将来、掲示板システムによって「最もコストパフォーマンスの良い製品の一つ」と判断されたオーディオプレーヤーを購入して、「あなたの今の気分にぴったりの曲リスト」オペレーションの結果として得られたプレイリストの曲を聞きながら、「今ある食材で作れるおすすめの夕食」を推薦してもらう、そんな生活スタイルが当たり前になる日が来ると思われる。

参考文献

[1] “add your own”, http://www.addyourown.com/.

[2] Apache Jakarta Project, “Tomcat,” http://jakarta.apache.org/tomcat/.

[3] “Apache Software Foundation”, http://www.apache.org/.

[4] Apache Software Foundation, “Apache,” http://httpd.apache.org/.

[5] Apache XML Project, “Apache Xindice,” http://xml.apache.org/xindice/.

[6] Apache XML Project, “Xalan,” http://xml.apache.org/xalan-j/.

[7] Apache XML Project, “Xerces,” http://xml.apache.org/xerces2-j/.

[8] ASAMI Tomoharu, “Relaxer Reference Manual,” http://www.relaxer.org/doc/refman/1.0/html/refman_en.html, 2003.

[9] “IIJ”, http://www.iij.ad.jp/.

[10] James Clark, “TREX - Tree Regular Expressions for XML,” http://www.thaiopensource.com/trex/.

[11] MURATA Makoto, “RELAX (Regular Language description for XML),” http://www.xml.gr.jp/relax/.

[12] OASIS, “RELAX NG Specification,” http://www.oasis-open.org/committees/relax-ng/spec.html.

[13] “OASIS”, http://www.oasis-open.org/home/index.php, 2001.

[14] PostgreSQL Global Development Group, “PostgreSQL,” http://www.postgresql.org/.

[15] RSS-DEV Working Group, “RDF Site Summary (RSS) 1.0,” http://purl.org/rss/1.0/.

[16] Sun Microsystems, “Java Servlet,” http://java.sun.com/products/servlet/.

[17] The Perl Foundation, “Perl,” http://www.perl.org/.

[18] W3C, “Extensible Markup Language (XML),” http://www.w3.org/TR/REC-xml/.

[19] W3C, “SVG,” http://www.w3.org/Graphics/SVG/.

[20] W3C, “XSL,” http://www.w3.org/Style/XSL/.

[21] “2ちゃんねる”, http://www.2ch.net/.

[22] 藤村滋, 松村真宏, 石塚満, “機械学習による電子掲示板からの評判情報抽出,” 情報処理学会第65回全国大会講演論文集 (4), pp.451-452, 2003.

[23] “価格.com”, http://www.kakaku.com/.

[24] “オリジナル☆ランキング”, http://guriuri.com/ranking/.

[25] 高橋 徹, 武田 英明, “TelMeA: 非同期コミュニティシステムにおけるAvatar-likeエージェントの効果とWebベースシステムへの実装,” 電子情報通信学会論文誌 D-I, Vol.J84-D-I, No.8, pp.1244-1255, 2001.

[26] 山西健司, “テキストマイニングとNLPビジネス,” 自然言語処理技術に関するシンポジウム, 2003.

付録

最後に、本研究で構築した電子掲示板システムの API ドキュメントを添付する。本システムは、以下に示すクラスから構成されている。

- AdminServlet
- AdminXMLPart
- BBSOperation
- BBSStructure
- BBSSubroutine
- CreateBoardServlet
- CreateBoardXMLPart
- CreateThreadServlet
- CreateThreadXMLPart
- EditMessageServlet
- EditMessageXMLPart
- ExecSystemCommand
- FileUploadServlet
- OperationServlet
- OperationXMLPart
- RebuildDBServlet
- SearchWordServlet
- SessionServlet
- UserAccountServlet
- WriteMessageServlet
- WriteMessageXMLPart
- XmlbbsFormatter

パッケージ com.semcode.xmlbbs

クラスの概要	
AdminServlet	管理者モードでの様々な設定を行うサーブレット
AdminXMLPart	管理者モードにおける様々なデータ操作を行う
BBSOperation	掲示板のオペレーションを管理する
BBSStructure	投稿構造を管理する
BBSSubroutine	サブルーチンの管理を行う
CreateBoardServlet	掲示板カテゴリを作成するサーブレット
CreateBoardXMLPart	掲示板カテゴリを作成する際のデータ操作
CreateThreadServlet	スレッドを作成するサーブレット
CreateThreadXMLPart	スレッドを作成する際のデータ操作を行う
EditMessageServlet	投稿の編集を行うサーブレット
EditMessageXMLPart	投稿の編集に関して実際にデータ操作を行う
ExecSystemCommand	システムコマンドを実行する
FileUploadServlet	ファイルアップロード全般を司る
OperationServlet	オペレーション・サブルーチン関連のユーザインタフェース
OperationXMLPart	オペレーション・サブルーチン関連でデータ操作を実際に行う
RebuildDBServlet	データベースの再構築を行う
SearchWordServlet	全文検索を行うサーブレット
SessionServlet	ログイン時のセッションを管理する
UserAccountServlet	ユーザアカウントの設定を行うサーブレット
WriteMessageServlet	投稿を行うサーブレット
WriteMessageXMLPart	投稿に関して実際にデータ操作を行う
XmlbbsFormatter	掲示板の閲覧のためのサーブレット

例外の概要

com.semcode.xmlbbs

クラス AdminServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.AdminServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **AdminServlet**
extends javax.servlet.http.HttpServlet

管理者モードでの様々な設定を行うサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

AdminServlet()

メソッドの概要

void	**doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	**doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

AdminServlet

```
public AdminServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス AdminXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.AdminXMLPart
```

public class **AdminXMLPart**
extends java.lang.Object

管理者モードにおける様々なデータ操作を行う

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

AdminXMLPart(java.lang.String iniFile)

メソッドの概要

boolean	**deleteAccount**(int num, java.lang.String pass) ユーザアカウントを削除する。
boolean	**deleteAccount**(java.lang.String id, java.lang.String pass) アカウントをデータベースから削除する。
boolean	**existSameID**(java.lang.String id, int except_num) ユーザがデータベース上に存在するかどうかを返す。
java.lang.String	**getBBSName**() 掲示板システム名を取得する。
java.lang.String	**getBGColor**() 掲示板システムの背景色を取得。
java.lang.String[]	**getBoardDescriptions**() 掲示板カテゴリの説明一覧を取得。
void	**getBoardInfo**() 掲示板情報をデータベースから読み出す。
java.lang.String[]	**getBoardNames**() 掲示板カテゴリ名の一覧を取得。
java.lang.String	**getCLColor**() 掲示板システムの見出しセルのを取得。
void	**getColors**() 掲示板システムの色情報を読み込む。
java.lang.String[]	**getUserIDs**() ユーザID一覧を取得。
java.lang.String[]	**getUserNames**()

	ユーザ名一覧を取得。
void	**getUsers**() ユーザ情報をデータベースから読み出す。
boolean	**login**(java.lang.String id, java.lang.String pass) ログインする。
boolean	**makeAccount**(java.lang.String id, java.lang.String name, java.lang.String password) アカウントをデータベースに作成する。
static java.lang.String	**MD5**(java.lang.String str) MD5による暗号化結果を返す。
boolean	**onepathlogin**(java.lang.String id, java.lang.String pass) ワンパスログインする。
boolean	**updateAccount**(int num, java.lang.String id, java.lang.String name, java.lang.String password_old, java.lang.String password_new) ユーザのアカウント情報を更新する。
boolean	**updateBoardInfo**(int num, java.lang.String name, java.lang.String description) 掲示板カテゴリ情報を更新する。
boolean	**UpdateSetting**(java.lang.String bbsname, java.lang.String bgcolor, java.lang.String clcolor) 掲示板システムの設定を変更する。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

AdminXMLPart

```
public AdminXMLPart(java.lang.String iniFile)
```

メソッドの詳細

UpdateSetting

```
public boolean UpdateSetting(java.lang.String bbsname,
                             java.lang.String bgcolor,
                             java.lang.String clcolor)
                      throws javax.xml.transform.TransformerException
```

掲示板システムの設定を変更する。

パラメータ:
- bbsname – 掲示板システム名
- bgcolor – 背景色
- clcolor – 見出しセルの背景色

戻り値:
成功true、失敗false

例外:
javax.xml.transform.TransformerException

---

getColors

```
public void getColors()
            throws java.io.IOException
```

掲示板システムの色情報を読み込む。実際に取得するときはgetBGColorメソッドとgetCLColorメソッドを使う。

例外:
`java.io.IOException`

---

makeAccount

```
public boolean makeAccount(java.lang.String id,
                           java.lang.String name,
                           java.lang.String password)
                    throws javax.xml.transform.TransformerException
```

アカウントをデータベースに作成する。

パラメータ:
`id` – ユーザID
`name` – フルネーム
`password` – パスワード
戻り値:
成功true、失敗false
例外:
`javax.xml.transform.TransformerException`

---

updateAccount

```
public boolean updateAccount(int num,
                             java.lang.String id,
                             java.lang.String name,
                             java.lang.String password_old,
                             java.lang.String password_new)
                      throws javax.xml.transform.TransformerException
```

ユーザのアカウント情報を更新する。

パラメータ:
`num` – ユーザの通し番号
`id` – ユーザID
`name` – ユーザ名
`password_old` – 古いパスワード
`password_new` – 新しいパスワード
戻り値:
成功true、失敗false
例外:
`javax.xml.transform.TransformerException`

---

updateBoardInfo

```
public boolean updateBoardInfo(int num,
                               java.lang.String name,
                               java.lang.String description)
```

throws javax.xml.transform.TransformerException

掲示板カテゴリ情報を更新する。

パラメータ:
num – 掲示板カテゴリの通し番号
name – 掲示板カテゴリ名
description – 掲示板カテゴリの説明
戻り値:
成功true、失敗false
例外:
javax.xml.transform.TransformerException

---

existSameID

```
public boolean existSameID(java.lang.String id,
                           int except_num)
                    throws javax.xml.transform.TransformerException
```

ユーザがデータベース上に存在するかどうかを返す。

パラメータ:
id – ユーザID
except_num – 除外するユーザの通し番号
戻り値:
存在するならtrue、存在しないならfalse
例外:
javax.xml.transform.TransformerException

---

login

```
public boolean login(java.lang.String id,
                     java.lang.String pass)
              throws javax.xml.transform.TransformerException
```

ログインする。

パラメータ:
id – ユーザID
pass – パスワード
戻り値:
成功true、失敗false
例外:
javax.xml.transform.TransformerException

---

onepathlogin

```
public boolean onepathlogin(java.lang.String id,
                            java.lang.String pass)
                     throws javax.xml.transform.TransformerException
```

ワンパスログインする。

パラメータ:
id – ユーザID

pass - パスワード

戻り値:

成功true、失敗false

例外:

javax.xml.transform.TransformerException

MD5

```
public static java.lang.String MD5(java.lang.String str)
```

MD5による暗号化結果を返す。

パラメータ:

str -

戻り値:

暗号化結果

---

deleteAccount

```
public boolean deleteAccount(java.lang.String id,
                             java.lang.String pass)
                      throws javax.xml.transform.TransformerException
```

アカウントをデータベースから削除する。

パラメータ:

id - ユーザID

pass - パスワード

戻り値:

成功true、失敗false

例外:

javax.xml.transform.TransformerException

---

deleteAccount

```
public boolean deleteAccount(int num,
                             java.lang.String pass)
                      throws javax.xml.transform.TransformerException
```

ユーザアカウントを削除する。

パラメータ:

num - ユーザの通し番号

pass - パスワード

戻り値:

成功true、失敗false

例外:

javax.xml.transform.TransformerException

---

getBBSName

```
public java.lang.String getBBSName()
                            throws java.io.IOException
```

掲示板システム名を取得する。

戻り値:
掲示板システム名
例外:
`java.io.IOException`

---

getUsers

```
public void getUsers()
            throws javax.xml.transform.TransformerException
```

ユーザ情報をデータベースから読み出す。この後、getUserNamesなどのメソッドによって取得。

例外:
`javax.xml.transform.TransformerException`

---

getBoardInfo

```
public void getBoardInfo()
                 throws javax.xml.transform.TransformerException
```

掲示板情報をデータベースから読み出す。この後、getBoardNamesなどのメソッドによって取得。

例外:
`javax.xml.transform.TransformerException`

---

getBGColor

```
public java.lang.String getBGColor()
```

掲示板システムの背景色を取得。前もってgetBoardInfoを実行しておく必要がある。

戻り値:

---

getCLColor

```
public java.lang.String getCLColor()
```

掲示板システムの見出しセルのを取得。前もってgetBoardInfoを実行しておく必要がある。

戻り値:

---

getUserNames

```
public java.lang.String[] getUserNames()
```

ユーザ名一覧を取得。前もってgetUsersを実行しておく必要がある。

戻り値:

---

getUserIDs

`public java.lang.String[] getUserIDs()`

ユーザID一覧を取得。前もってgetUsersを実行しておく必要がある。

戻り値:

---

getBoardNames

`public java.lang.String[] getBoardNames()`

掲示板カテゴリ名の一覧を取得。前もってgetBoardInfoを実行しておく必要がある。

戻り値:

---

getBoardDescriptions

`public java.lang.String[] getBoardDescriptions()`

掲示板カテゴリの説明一覧を取得。前もってgetBoardInfoを実行しておく必要がある。

戻り値:

---

com.semcode.xmlbbs

クラス BBSOperation

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.BBSOperation
```

public class **BBSOperation**
extends java.lang.Object

掲示板のオペレーションを管理する

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

BBSOperation()

メソッドの概要

boolean	**addOperation**(java.lang.String iniFile, java.lang.String name, java.lang.String description, java.lang.String targetclass, java.lang.String script, java.lang.String path) オペレーションをデータベースに追加する。
java.lang.String[]	**getClass**(int n) n番目のオペレーションのクラス名を取得。
java.lang.String	**getDescription**(int n) n番目のオペレーションの説明を取得。
int	**getElementCount**() オペレーションの数を取得。
java.lang.String	**getFile**(int n) n番目のオペレーションの保存されたファイル名を取得。
java.lang.String	**getName**(int n) n番目のオペレーションの名前を取得。
void	**getOperation**(java.lang.String iniFile) iniFileを読み込んで登録されているオペレーションの情報を取得する。

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

BBSOperation

```
public BBSOperation()
```

メソッドの詳細

getOperation

```
public void getOperation(java.lang.String iniFile)
                  throws java.io.IOException
```

iniFileを読み込んで登録されているオペレーションの情報を取得する。その結果はgetNameやgetClassなどのメソッドを用いて得る。

パラメータ:
iniFile - iniファイル
例外:
java.io.IOException

---

addOperation

```
public boolean addOperation(java.lang.String iniFile,
                            java.lang.String name,
                            java.lang.String description,
                            java.lang.String targetclass,
                            java.lang.String script,
                            java.lang.String path)
                     throws java.io.IOException
```

オペレーションをデータベースに追加する。

パラメータ:
iniFile - iniファイル
name - オペレーション名
description - オペレーションの説明
targetclass - 対象クラス
script - スクリプト
path - 保存するパス名
戻り値:
成功true、失敗false
例外:
java.io.IOException

---

getName

```
public java.lang.String getName(int n)
```

n番目のオペレーションの名前を取得。

パラメータ:
n -
戻り値:
オペレーションの名前

---

getDescription

```
public java.lang.String getDescription(int n)
```

n番目のオペレーションの説明を取得。

パラメータ:
n –
戻り値:
オペレーションの説明

---

getFile

```
public java.lang.String getFile(int n)
```

n番目のオペレーションの保存されたファイル名を取得。

パラメータ:
n –
戻り値:
オペレーションの保存されたファイル名

---

getClass

```
public java.lang.String[] getClass(int n)
```

n番目のオペレーションのクラス名を取得。

パラメータ:
n –
戻り値:
オペレーションのクラス名

---

getElementCount

```
public int getElementCount()
```

オペレーションの数を取得。

戻り値:
オペレーションの数

---

com.semcode.xmlbbs

クラス BBSStructure

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.BBSStructure
```

public class **BBSStructure**
extends java.lang.Object

投稿構造を管理する

作成者:
MATSUMOTO Kazuyuki

メソッドの概要

int	**getElementCount**() プロパティ数を取得。
java.lang.String	**getName**(int n) n番目プロパティの名前を取得。
boolean	**getOptional**(int n) n番目プロパティが省略可能な要素かどうかを取得。
java.lang.String	**getType**(int n) n番目プロパティの型を取得。
java.lang.String	**getValue**(int n) n番目プロパティの値を取得。
void	**setName**(int n, java.lang.String value) n番目のプロパティの名前としてvalueを設定する。
void	**setOptional**(int n, boolean value) n番目のプロパティが省略可能かどうかをvalueで設定する。
void	**setType**(int n, java.lang.String value) n番目のプロパティの型としてvalueを設定する。
void	**setValue**(int n, java.lang.String value) n番目のプロパティの値としてvalueを設定する。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

メソッドの詳細

getElementCount

```
public int getElementCount()
```

プロパティ数を取得。

戻り値:
プロパティ数

---

getName

```
public java.lang.String getName(int n)
```

n番目プロパティの名前を取得。

パラメータ:
n –
戻り値:

---

getType

```
public java.lang.String getType(int n)
```

n番目プロパティの型を取得。

パラメータ:
n –
戻り値:

---

getValue

```
public java.lang.String getValue(int n)
```

n番目プロパティの値を取得。

パラメータ:
n –
戻り値:

---

getOptional

```
public boolean getOptional(int n)
```

n番目プロパティが省略可能な要素かどうかを取得。

パラメータ:
n –
戻り値:

---

setName

```
public void setName(int n,
                    java.lang.String value)
```

n番目のプロパティの名前としてvalueを設定する。

パラメータ:
n –
value –

---

setType

```
public void setType(int n,
                    java.lang.String value)
```

n番目のプロパティの型としてvalueを設定する。

パラメータ:
n –
value –

---

setValue

```
public void setValue(int n,
                     java.lang.String value)
```

n番目のプロパティの値としてvalueを設定する。

パラメータ:
n –
value –

---

setOptional

```
public void setOptional(int n,
                        boolean value)
```

n番目のプロパティが省略可能かどうかをvalueで設定する。

パラメータ:
n –
value –

---

com.semcode.xmlbbs

クラス BBSSubroutine

java.lang.Object
  |
  +--com.semcode.xmlbbs.BBSSubroutine

---

public class **BBSSubroutine**
extends java.lang.Object

サブルーチンの管理を行う

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

BBSSubroutine()

メソッドの概要

boolean	addSubroutine(java.lang.String iniFile, java.lang.String name, java.lang.String description, java.lang.String parameter, java.lang.String targetclass, java.lang.String script, java.lang.String path) サブルーチンをデータベースに追加する。
java.lang.String[]	getClass(int n) n番目のサブルーチンのクラス名を取得。
java.lang.String	getDescription(int n) n番目のサブルーチンの説明を取得。
int	getElementCount() サブルーチンの数を取得。
java.lang.String	getFile(int n) n番目のサブルーチンのファイル名を取得。
java.lang.String	getName(int n) n番目のサブルーチンの名前を取得。
int	getParameter(int n) n番目のサブルーチンの引数の数を取得。
void	getSubroutine(java.lang.String iniFile) サブルーチン情報をデータベースから読み込む。

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

BBSSubroutine

```
public BBSSubroutine()
```

メソッドの詳細

getSubroutine

```
public void getSubroutine(java.lang.String iniFile)
                   throws java.io.IOException
```

サブルーチン情報をデータベースから読み込む。各情報はgetNameなどのメソッドを用いて取得する。

パラメータ:
iniFile - iniファイル
例外:
java.io.IOException

---

addSubroutine

```
public boolean addSubroutine(java.lang.String iniFile,
                             java.lang.String name,
                             java.lang.String description,
                             java.lang.String parameter,
                             java.lang.String targetclass,
                             java.lang.String script,
                             java.lang.String path)
                      throws java.io.IOException
```

サブルーチンをデータベースに追加する。

パラメータ:
iniFile - iniファイル
name - サブルーチン名
description - サブルーチンの説明
parameter - 引数の数
targetclass - 対象クラス
script - スクリプト
path - スクリプトのパス名
戻り値:
成功true、失敗false
例外:
java.io.IOException

---

getName

```
public java.lang.String getName(int n)
```

n番目のサブルーチンの名前を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

パラメータ:

n - サブルーチンの通し番号
戻り値:

---

getDescription

public java.lang.String **getDescription**(int n)

n番目のサブルーチンの説明を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

パラメータ:
n - サブルーチンの通し番号
戻り値:

---

getFile

public java.lang.String **getFile**(int n)

n番目のサブルーチンのファイル名を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

パラメータ:
n - サブルーチンの通し番号
戻り値:

---

getParameter

public int **getParameter**(int n)

n番目のサブルーチンの引数の数を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

パラメータ:
n - サブルーチンの通し番号
戻り値:

---

getClass

public java.lang.String[] **getClass**(int n)

n番目のサブルーチンのクラス名を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

パラメータ:
n - サブルーチンの通し番号
戻り値:

---

getElementCount

public int **getElementCount**()

サブルーチンの数を取得。前もってgetSubroutineメソッドを呼び出しておく必要がある。

戻り値:

---

com.semcode.xmlbbs

クラス CreateBoardServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.CreateBoardServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **CreateBoardServlet**
extends javax.servlet.http.HttpServlet

掲示板カテゴリを作成するサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

CreateBoardServlet()

メソッドの概要

void	**doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	**doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

CreateBoardServlet

```
public CreateBoardServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス CreateBoardXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.CreateBoardXMLPart
```

public class **CreateBoardXMLPart**
extends java.lang.Object

掲示板カテゴリを作成する際のデータ操作

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

CreateBoardXMLPart(java.lang.String iniFile)

メソッドの概要

boolean	**createBoard**(java.lang.String creater_id, java.lang.String title, java.lang.String description) 掲示板カテゴリを新規に作成する。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

CreateBoardXMLPart

```
public CreateBoardXMLPart(java.lang.String iniFile)
```

パラメータ:
iniFile – iniファイル

メソッドの詳細

createBoard

```
public boolean createBoard(java.lang.String creater_id,
                           java.lang.String title,
                           java.lang.String description)
```

掲示板カテゴリを新規に作成する。

パラメータ:

`creater_id` – 作成者
`title` – 掲示板カテゴリ名
`description` – 掲示板カテゴリの説明

戻り値:

---

com.semcode.xmlbbs

クラス CreateThreadServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.CreateThreadServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **CreateThreadServlet**
extends javax.servlet.http.HttpServlet

スレッドを作成するサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

CreateThreadServlet()

メソッドの概要

void	**doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	**doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

CreateThreadServlet

```
public CreateThreadServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス CreateThreadXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.CreateThreadXMLPart
```

public class **CreateThreadXMLPart**
extends java.lang.Object

スレッドを作成する際のデータ操作を行う

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

CreateThreadXMLPart(int boardid, java.lang.String iniFile, java.lang.String servletRoot)

メソッドの概要

boolean	**createThread**(java.lang.String creater_id, java.lang.String thread_title, org.w3c.dom.Document content, java.lang.String messageClass, java.lang.String restype, java.lang.String filename, java.lang.String host) 新規にスレッドを作成する。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

CreateThreadXMLPart

```
public CreateThreadXMLPart(int boardid,
                           java.lang.String iniFile,
                           java.lang.String servletRoot)
```

メソッドの詳細

createThread

```
public boolean createThread(java.lang.String creater_id,
                            java.lang.String thread_title,
                            org.w3c.dom.Document content,
                            java.lang.String messageClass,
                            java.lang.String restype,
```

```
                    java.lang.String filename,
                    java.lang.String host)
```

新規にスレッドを作成する。

パラメータ:

- `creater_id` – 作成者ID
- `thread_title` – スレッド名
- `content` – 投稿本文
- `messageClass` – クラス名
- `restype` – 返信クラス名
- `filename` – 添付ファイル名
- `host` – ホスト情報

戻り値:

成功true、失敗false

---

com.semcode.xmlbbs

クラス EditMessageServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.EditMessageServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **EditMessageServlet**
extends javax.servlet.http.HttpServlet

投稿の編集を行うサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

EditMessageServlet()

メソッドの概要

void	doGet(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

EditMessageServlet

```
public EditMessageServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス EditMessageXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.EditMessageXMLPart
```

public class **EditMessageXMLPart**
extends java.lang.Object

投稿の編集に関して実際にデータ操作を行う

作成者:
MATSUMOTO Kazuyuki

メソッドの概要

boolean	**deleteMessage**(java.lang.String bid, java.lang.String tid, int mid) 投稿を削除する。
boolean	**editMessage**(java.lang.String bid, java.lang.String tid, int mid, org.w3c.dom.Document doc) 投稿を削除する。
java.lang.String	**getMessageText**(java.lang.String bid, java.lang.String tid, java.lang.String mid) 投稿本文の取得。
java.lang.String	**getMessageWriter**(java.lang.String bid, java.lang.String tid, java.lang.String mid) 投稿者名の取得。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

メソッドの詳細

deleteMessage

```
public boolean deleteMessage(java.lang.String bid,
                             java.lang.String tid,
                             int mid)
                      throws javax.xml.transform.TransformerException
```

投稿を削除する。

パラメータ:
bid - 掲示板カテゴリID
tid - スレッドID
mid - 投稿ID

戻り値:
成功true、失敗false
例外:
javax.xml.transform.TransformerException

---

getMessageText

```
public java.lang.String getMessageText(java.lang.String bid,
                                       java.lang.String tid,
                                       java.lang.String mid)
```

投稿本文の取得。

パラメータ:
bid – 掲示板カテゴリID
tid – スレッドID
mid – 投稿ID
戻り値:
投稿本文

---

getMessageWriter

```
public java.lang.String getMessageWriter(java.lang.String bid,
                                         java.lang.String tid,
                                         java.lang.String mid)
```

投稿者名の取得。

パラメータ:
bid – 掲示板カテゴリID
tid – スレッドID
mid – 投稿ID
戻り値:
投稿者名

---

editMessage

```
public boolean editMessage(java.lang.String bid,
                           java.lang.String tid,
                           int mid,
                           org.w3c.dom.Document doc)
```

投稿を削除する。

パラメータ:
bid – 掲示板カテゴリID
tid – スレッドID
mid – 投稿ID
doc – 投稿本体
戻り値:
成功true、失敗false
例外:
javax.xml.transform.TransformerException

---

com.semcode.xmlbbs

クラス ExecSystemCommand

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.ExecSystemCommand
```

public class **ExecSystemCommand**
extends java.lang.Object

システムコマンドを実行する

作成者:
MATSUMOTO Kazuyuki

コンストラクタの概要

ExecSystemCommand()

メソッドの概要

java.lang.String[]	**execPerlScript**(java.io.File file, java.lang.String varinfo) Perlで書かれたファイルを実行する。
java.lang.String[]	**execPerlScript**(java.lang.String script, java.lang.String varinfo) Perlで書かれたスクリプトを実行する。
void	**setDataFolder**(java.lang.String folder) perlスクリプトのフォルダ名をセットする。

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

ExecSystemCommand

```
public ExecSystemCommand()
```

メソッドの詳細

execPerlScript

```
public java.lang.String[] execPerlScript(java.io.File file,
                                         java.lang.String varinfo)
```

Perlで書かれたファイルを実行する。

パラメータ:
`file` – ファイル
`varinfo` – 変数情報
戻り値:
出力結果。行ごとに分かれた配列。

---

execPerlScript

```
public java.lang.String[] execPerlScript(java.lang.String script,
                                         java.lang.String varinfo)
```

Perlで書かれたスクリプトを実行する。

パラメータ:
`script` – スクリプト
`varinfo` – 変数情報
戻り値:
出力結果。行ごとに分かれた配列。

---

setDataFolder

```
public void setDataFolder(java.lang.String folder)
```

perlスクリプトのフォルダ名をセットする。

パラメータ:
`folder` – フォルダ名

---

com.semcode.xmlbbs

クラス FileUploadServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.FileUploadServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **FileUploadServlet**
extends javax.servlet.http.HttpServlet

ファイルアップロード全般を司る

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

FileUploadServlet()

メソッドの概要

void	doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
void	init()

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

コンストラクタの詳細

FileUploadServlet

public **FileUploadServlet**()

メソッドの詳細

init

public void **init**()
throws javax.servlet.ServletException

オーバーライド:
クラス javax.servlet.GenericServlet 内の init
javax.servlet.ServletException

---

doPost

public void **doPost**(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response)
throws javax.servlet.ServletException,
java.io.IOException

オーバーライド:
クラス javax.servlet.http.HttpServlet 内の doPost
javax.servlet.ServletException
java.io.IOException

---

com.semcode.xmlbbs

クラス OperationServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.OperationServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **OperationServlet**
extends javax.servlet.http.HttpServlet

オペレーション・サブルーチン関連のユーザインタフェース

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

OperationServlet()

メソッドの概要

void	doGet(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

OperationServlet

```
public OperationServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス OperationXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.OperationXMLPart
```

public class **OperationXMLPart**
extends java.lang.Object

オペレーション・サブルーチン関連でデータ操作を実際に行う

作成者:
MATSUMOTO Kazuyuki

メソッドの概要

static BBSStructure	**getBBSStructureByClassName**(java.lang.String className, java.lang.String iniFile) クラス名からBBSStructureクラスを返す。
org.w3c.dom.Document	**getDocumentInACondition**(java.lang.String bid, java.lang.String tid, java.lang.String className, java.lang.String[] article, java.lang.String[] value, java.lang.String[] operator, int nCondition, java.lang.String order) 指定された条件に合う投稿のDocumentを返す。
java.lang.String	**getVarStringByDocument**(org.w3c.dom.Document doc) 投稿一覧のDocumentから、各プロパティ値のperl用変数情報を取得する。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

メソッドの詳細

getDocumentInACondition

```
public org.w3c.dom.Document getDocumentInACondition(java.lang.String bid,
                                                    java.lang.String tid,
                                                    java.lang.String className,
                                                    java.lang.String[] article,
                                                    java.lang.String[] value,
                                                    java.lang.String[] operator,
                                                    int nCondition,
                                                    java.lang.String order)
```

指定された条件に合う投稿のDocumentを返す。

パラメータ:

bid – 掲示板カテゴリID
tid – スレッドID
className – クラス名
article – プロパティの配列
value – パラメータの配列
operator – 条件オペレータの配列
nCondition – 条件の数
order – 表示順序

戻り値:

条件を満たす投稿のDOM

---

getBBSStructureByClassName

```
public static BBSStructure getBBSStructureByClassName(java.lang.String className,
                                                      java.lang.String iniFile)
                                               throws java.io.IOException
```

クラス名からBBSStructureクラスを返す。

パラメータ:

className – クラス名
iniFile – iniファイル

戻り値:

投稿構造

例外:

java.io.IOException

---

getVarStringByDocument

```
public java.lang.String getVarStringByDocument(org.w3c.dom.Document doc)
```

投稿一覧のDocumentから、各プロパティ値のperl用変数情報を取得する。

パラメータ:

doc – 投稿一覧のDocument

戻り値:

perl用変数情報

---

com.semcode.xmlbbs

クラス RebuildDBServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.RebuildDBServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **RebuildDBServlet**
extends javax.servlet.http.HttpServlet

データベースの再構築を行う

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

RebuildDBServlet()

メソッドの概要

void	doGet(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

RebuildDBServlet

```
public RebuildDBServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス SearchWordServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.SearchWordServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **SearchWordServlet**
extends javax.servlet.http.HttpServlet

全文検索を行うサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

SearchWordServlet()

メソッドの概要

void	doGet(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

SearchWordServlet

```
public SearchWordServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス SessionServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.SessionServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **SessionServlet**
extends javax.servlet.http.HttpServlet

ログイン時のセッションを管理する

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

SessionServlet()

メソッドの概要

void	doGet(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	doPost(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

SessionServlet

```
public SessionServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス UserAccountServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.UserAccountServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **UserAccountServlet**
extends javax.servlet.http.HttpServlet

ユーザアカウントの設定を行うサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

UserAccountServlet()

メソッドの概要

void	**doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	**doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

UserAccountServlet

```
public UserAccountServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス WriteMessageServlet

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.WriteMessageServlet
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **WriteMessageServlet**
extends javax.servlet.http.HttpServlet

投稿を行うサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

WriteMessageServlet()

メソッドの概要

void	**doGet**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
void	**doPost**(javax.servlet.http.HttpServletRequest req, javax.servlet.http.HttpServletResponse res)
static java.lang.String	**getClassList**() ブラウザ用のクラスリストを取得

クラス javax.servlet.http.HttpServlet から継承したメソッド

service

クラス javax.servlet.GenericServlet から継承したメソッド

destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

WriteMessageServlet

```
public WriteMessageServlet()
```

メソッドの詳細

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest req,
                   javax.servlet.http.HttpServletResponse res)
            throws javax.servlet.ServletException,
                   java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doPost

javax.servlet.ServletException

java.io.IOException

---

getClassList

```
public static java.lang.String getClassList()
```

ブラウザ用のクラスリストを取得

戻り値:

クラスリスト

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest req,
                  javax.servlet.http.HttpServletResponse res)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

オーバーライド:

クラス javax.servlet.http.HttpServlet 内の doGet

javax.servlet.ServletException

java.io.IOException

---

com.semcode.xmlbbs

クラス WriteMessageXMLPart

```
java.lang.Object
  |
  +--com.semcode.xmlbbs.WriteMessageXMLPart
```

public class **WriteMessageXMLPart**
extends java.lang.Object

投稿に関して実際にデータ操作を行う

作成者:
MATSUMOTO Kazuyuki

入れ子クラスの概要

class	WriteMessageXMLPart.UncompleteFormException

コンストラクタの概要

WriteMessageXMLPart(java.lang.String iniFile)
WriteMessageXMLPart(java.lang.String boardid, java.lang.String threadid, java.lang.String iniFile, java.lang.String servletRoot)

メソッドの概要

static java.lang.String	getBBSName(java.lang.String iniFile) 掲示板システム名を取得する。
static java.lang.String	getBoardTitle(java.lang.String bid, java.lang.String iniFile) 掲示板カテゴリ名を取得する。
static java.lang.String[]	getClassList(java.lang.String bid, java.lang.String tid, java.lang.String iniFile) 指定したスレッドに含まれる投稿クラスのリストを返す。
org.w3c.dom.Document	getContentDOM(java.lang.String[] componentValue, java.lang.String messageClass) クラス名と各プロパティの値から、投稿内容を表すDOMを取得。
org.w3c.dom.Document	getDocument(java.lang.String key) キー名からDocumentを取得する。
static java.lang.String	getMessageClass(java.lang.String bid, java.lang.String tid, int res, java.lang.String iniFile) 指定した投稿に対する返信の投稿クラスを取得する。
java.lang.String	getMessageTitle(java.lang.String bid, java.lang.String tid, java.lang.String mid)

	投稿のタイトルを取得。
static java.lang.String	**getMessageTitle**(java.lang.String bid, java.lang.String tid, java.lang.String mid, java.lang.String iniFile) 投稿のタイトルを取得。
static java.lang.String	**getNameByID**(java.lang.String id, java.lang.String iniFile) ユーザIDからフルネームを取得する
static java.lang.String	**getThreadTitle**(java.lang.String bid, java.lang.String tid, java.lang.String iniFile) スレッド名を取得する。
boolean	**writeMessage**(java.lang.String name, org.w3c.dom.Document content, java.lang.String messageClass, java.lang.String restype, java.lang.String filename, int res, java.lang.String host) 新規投稿をデータベースに書き込む。

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

WriteMessageXMLPart

```
public WriteMessageXMLPart(java.lang.String boardid,
                           java.lang.String threadid,
                           java.lang.String iniFile,
                           java.lang.String servletRoot)
```

パラメータ:

- boardid – 掲示板カテゴリID
- threadid – スレッドID
- iniFile – iniファイル
- servletRoot – サーブレットのルートディレクトリ情報

WriteMessageXMLPart

```
public WriteMessageXMLPart(java.lang.String iniFile)
```

パラメータ:

- iniFile – iniファイル

メソッドの詳細

writeMessage

```
public boolean writeMessage(java.lang.String name,
                            org.w3c.dom.Document content,
                            java.lang.String messageClass,
                            java.lang.String restype,
                            java.lang.String filename,
                            int res,
                            java.lang.String host)
                     throws org.w3c.dom.DOMException,
                            javax.xml.transform.TransformerException,
                            java.io.IOException
```

新規投稿をデータベースに書き込む。

パラメータ:
name - 投稿者名
content - 投稿内容
messageClass - クラス
restype - 返信クラス
filename - 添付ファイル
res - 返信先
host - ホスト情報
戻り値:
成功true、失敗false
例外:
org.w3c.dom.DOMException
javax.xml.transform.TransformerException
java.io.IOException

---

getBoardTitle

```
public static java.lang.String getBoardTitle(java.lang.String bid,
                                              java.lang.String iniFile)
                                       throws java.io.IOException
```

掲示板カテゴリ名を取得する。

パラメータ:
bid - 掲示板カテゴリID
iniFile - iniファイル
戻り値:
カテゴリ名
例外:
java.io.IOException

---

getThreadTitle

```
public static java.lang.String getThreadTitle(java.lang.String bid,
                                               java.lang.String tid,
                                               java.lang.String iniFile)
                                        throws java.io.IOException
```

スレッド名を取得する。

パラメータ:
bid - 掲示板カテゴリID
tid - スレッドID
iniFile - iniファイル
戻り値:
スレッド名
例外:
java.io.IOException

---

getMessageTitle

```
public static java.lang.String getMessageTitle(java.lang.String bid,
                                                java.lang.String tid,
                                                java.lang.String mid,
```

```
                                        java.lang.String iniFile)
                                 throws java.io.IOException
```

投稿のタイトルを取得。

パラメータ:
bid - 掲示板カテゴリID
tid - スレッドID
mid - 投稿ID
iniFile - iniファイル
戻り値:
タイトル
例外:
java.io.IOException

---

getMessageTitle

```
public java.lang.String getMessageTitle(java.lang.String bid,
                                        java.lang.String tid,
                                        java.lang.String mid)
                                 throws java.io.IOException
```

投稿のタイトルを取得。

パラメータ:
bid - 掲示板カテゴリID
tid - スレッドID
mid - 投稿ID
戻り値:
タイトル
例外:
java.io.IOException

---

getBBSName

```
public static java.lang.String getBBSName(java.lang.String iniFile)
                                   throws java.io.IOException
```

掲示板システム名を取得する。

パラメータ:
iniFile - iniファイル
戻り値:
掲示板名
例外:
java.io.IOException

---

getNameByID

```
public static java.lang.String getNameByID(java.lang.String id,
                                           java.lang.String iniFile)
                                    throws java.io.IOException
```

ユーザIDからフルネームを取得する

パラメータ:

id – ユーザID
iniFile – iniファイル

戻り値:
フルネーム

例外:
java.io.IOException

---

getDocument

```
public org.w3c.dom.Document getDocument(java.lang.String key)
```

キー名からDocumentを取得する。

パラメータ:
key – データベースのキー名

戻り値:
DOM

---

getContentDOM

```
public org.w3c.dom.Document getContentDOM(java.lang.String[] componentValue,
                                          java.lang.String messageClass)
                                   throws WriteMessageXMLPart.UncompleteFormException
```

クラス名と各プロパティの値から、投稿内容を表すDOMを取得。

パラメータ:
componentValue – 各プロパティ値
messageClass – クラス名

戻り値:
DOM

例外:
WriteMessageXMLPart.UncompleteFormException – フォームが不完全

---

getMessageClass

```
public static java.lang.String getMessageClass(java.lang.String bid,
                                                java.lang.String tid,
                                                int res,
                                                java.lang.String iniFile)
                                         throws java.io.IOException
```

指定した投稿に対する返信の投稿クラスを取得する。

パラメータ:
bid – 掲示板カテゴリID
tid – スレッドID
res – 投稿番号
iniFile – iniファイル

戻り値:
投稿クラス

例外:
java.io.IOException

---

getClassList

```
public static java.lang.String[] getClassList(java.lang.String bid,
                                               java.lang.String tid,
                                               java.lang.String iniFile)
                                        throws java.io.IOException
```

指定したスレッドに含まれる投稿クラスのリストを返す。

パラメータ:

`bid` – 掲示板カテゴリID
`tid` – スレッドID
`iniFile` – iniファイル

戻り値:

投稿クラスのリスト

例外:

`java.io.IOException`

---

com.semcode.xmlbbs

クラス XmlbbsFormatter

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet
        |
        +--javax.servlet.http.HttpServlet
              |
              +--com.semcode.xmlbbs.XmlbbsFormatter
```

すべての実装インタフェース:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

public class **XmlbbsFormatter**
extends javax.servlet.http.HttpServlet

掲示板の閲覧のためのサーブレット

作成者:
MATSUMOTO Kazuyuki
関連項目:
直列化された形式

コンストラクタの概要

XmlbbsFormatter()

クラス javax.servlet.http.HttpServlet から継承したメソッド
service

クラス javax.servlet.GenericServlet から継承したメソッド
destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

クラス java.lang.Object から継承したメソッド
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

コンストラクタの詳細

XmlbbsFormatter

public **XmlbbsFormatter**()

謝辞

本研究を行うにあたり日頃御指導を賜わった、 長尾確 教授、大平茂輝 助手には研究の基礎的な考え方や進め方、論文の指導など大変お世話になりました。またゼミ等においても数々の貴重な意見を頂きました。御礼申し上げます。

また、研究室のメンバーである友部博教さん、梶克彦さん、山本大介くん、小酒井一稔くん、本村可奈子さん、佐橋典幸くん、土田貴裕くんには、ゼミ等で様々な意見を頂いたり、実装において分からないことがあったりしたときに相談したり、また生活の面でも色々お世話になりました。また秘書の金子幸子さんには、学生生活や研究活動のための様々なサポートを頂きました。

本論文が完成したのは、卒業された方も含めて、長尾研究室の全ての仲間のおかげであると思っています。この場を借りて御礼申し上げます。