2007年度修士論文

ウェブサイト構築を目的とした データフロー型ビジュアルプログラミング言語 「ゆば」の開発

Yuba, a dataflow-oriented visual programming language for web development

2008年2月4日 提出

指導：筧捷彦教授

早稲田大学大学院理工学研究科情報ネットワーク専攻
学籍番号：3606U103-9

三浦 琢磨

目次

第1章　序論

著者らは今回新しい言語処理系を開発し，「ゆば」と命名した。

ゆばは，ウェブサービスの構築に目的を特化したビジュアルプログラミング言語処理系であり，言語体系そのものと，その開発環境・実行環境の総称である。

著者らがゆばの開発を目指すことになった背景から述べる。

1.1　オンラインコミュニケーションの隆盛

インターネットの利用は，ブラウザの登場，WWW の普及による試行錯誤的な時期から一段落してきた。近年はネットワークの特性を活かした，より実用的な，もしくは新しい社会的価値を生むサービスが続々と提供され，それらは俗に Web2.0 と総称されている[1]。

Web2.0 に整理されるサービス群に見られる特徴は，ユーザが集まってコンテンツを発信しあうことがより大きな付加価値を持つコンテンツを形成する CGM（Consumer Generated Media）であるという点が挙げられる。

まず，現在どのようなサービスが Web2.0 として注目されているかを概観しよう。

1.1.1　blog

blog は日記のような系時的に書きためるテキスト群を管理・掲示するサービスである。記事そのものも目次ページも，ユーザが HTML で書き下ろす必要はない。サイト上で記事を打ち込むだけで自動的に記事ページが作成されて固有の URL[1]が付与され，目次ページや一覧表示ページ，記事検索サービスも動的に提供される。

blog の特徴は記事を管理することだけではない。現在の blog システムは**トラックバック・RSS 配信**という，記事の再利用性を高めるふたつの仕組みを実装していることが多い。

トラックバックは，記事中で他 blog の記事に言及したことが言及先 blog へ通知され[2]，言及先の記事にも言及元の記事への逆参照を表示できるようになるものである。blog 越しに形成される議論のネットワークが追跡できるようになるのだ。

RSS は，記事の一覧を提供するための交換形式である。RSS リーダによって blog の購読が自動化できるだけでなく，複数の RSS を一本の RSS にまとめたり[3]，その中から検索結果に該当する記事だけを配信するようなサービスも可能であり，記事は複数のサービスが形成するネットワーク上で様々な形で提示されるようになる

1.1.2　SNS

SNS（Social Networking Service）は会員制サイトであり，会員同士の友人関係を電子的にネッ

[1] permalink と呼ぶ.

[2] トラックバック ping と呼ぶ。

[3] これらのサービスはアグリゲータと呼ばれる。

トワーク化するサービスである[1]。会員は自分用のページにプロフィールや写真を掲示し，日記や掲示板などの会員向けサービスを利用できる。

大きな特徴は会員同士が知り合いの会員を自分の友人だと登録できるようにしていることである。友人だと登録することで，友人すべての日記の更新を一覧したり，友人だけに公開する日記を書くなどできるようになる。

1.1.3 WikiWikiWeb

WikiWikiWeb（以下，Wiki と略する）は，閲覧者が誰でも編集できるウェブサイトを構築するシステムである[2]。通常のウェブサイトは HTML で記述され，設置者だけがファイルをアップロードすることで更新できる。しかし Wiki で構築するサイトでは，HTML ファイルをアップロードするのではなく，サイト上で編集機能を呼び出し，文章を打ち込むことでページを新設したり更新したりする。編集に際して HTML タグを打ち込む必要はなく，Wiki 専用の簡易タグを使って文書の整形ができる。

編集できる人を制限することも可能だが，サイトの閲覧者の誰もが編集機能を使うことができる。結果として，編集作業を共有することができる。Wiki の応用としては，誰でも編集できる百科事典を目指した Wikipedia[2]が成功を収めているのがよく知られている。

1.1.4 検索エンジン

インターネット上のドキュメントをキーワード検索するための検索エンジンでは，ウェブページの重要度を決定するのに，ウェブページ群のリンク関係を利用するのが現在一般的である。Google[3]がはじめたこの手法によって，ウェブサイト制作者すべてが，ページ重要度ランキングの決定に参加することになった。

1.1.5 動画投稿サイト

利用者が投稿した動画ファイルを公開するサービスである[4]。公開された動画はブラウザのみで視聴できる。動画データの再生には，多くのブラウザに組み込まれた Flash プラグインを利用する。

動画投稿サイトは単に公開するだけでなく，Web2.0 的な機能を数多く持つ。ある投稿者のメンバーページからその人の投稿作品を一覧できたり，動画にコメントした人のメンバーページも見られたり，投稿された画像にタグと呼ばれるキーワード群を設定することで関連の強い動画を集約できたり，指定したタグを持つ新規動画を RSS 配信したりなどである。

さらに，新種の動画投稿サービスでは動画に重ねて字幕のようにコメントを投稿するという機能もあり，他の利用者と同時に試聴しながらコメントしあっているかのような感覚を擬似的に実現している[5]。

---

[1] "MySpace" http://www.myspace.com/, "mixi" http://mixi.jp/

[2] http://www.wikipedia.org/

[3] http://www.google.com/

[4] "YouTube" http://www.youtube.com/

[5] "ニコニコ動画" http://www.nicovideo.jp/

1.1.6　ソーシャルカレンダー

予定表を電子化するサイトである[1]。予定表を作るだけでなく，それを公開したり，他の利用者と共有したり，予定を RSS 配信したりすることができる。例えばスポーツのファンが試合の日程表を作って公開すると，他のファンがそれを自分のカレンダーに取り込むことで，自分の予定と並べて閲覧するようなことが可能になる。

1.1.7　ショッピングサイト

オンラインショッピングはインターネットが商用化してすぐに登場した。Web2.0 的ショッピングサイトは初期のショッピングサイトの機能に加え，顧客の購入履歴や商品閲覧履歴をサイト構成に利用することで新種のサービスとなっている[2]。

これら履歴をデータマイニング[3]にかけ,顧客それぞれに興味を持ちそうな商品を提案したり,ある商品のページにそれと関連の強い（同時に購入されることが多い）商品を提示したりといった販売促進ができる。顧客は商品を探したり購入したりしているだけのつもりでいても，人気ランキングや商品同士の関連づけに参加していることになるのである。

1.1.8　ミニブログ

blog は比較的長文で意見を書き，言及とトラックバックの鎖で議論を形成するものだが，もっと手軽に発言できるようにしたいと作られ，人気を呼んでいるのがミニブログである[3]。記事の文字数が制限されており，ひとこと日記だけを書ける。他の利用者の日記を取り込むよう登録しておくと，登録された利用者のひとこと日記も自分の日記の時系列一覧に加わって表示される。この機能によって，直接話しかけるわけでもなく，利用者同士で緩いコミュニケーションが発生するというものである。

1.2　ウェブサービスにおける利用と提供の不均衡

Web2.0 のモダンなウェブサービスにおいて，利用することは発信することとなっている。ブログや掲示板，SNS，動画投稿サイトのように能動的な発信そのものを目的としたサービスはもちろんのこと，ショッピングサイトや検索サイトのように利用するだけのサイトでもランキングに参加することで結果的に発信をし，付加価値を持つコンテンツを生成することになっている。

サービスを利用するのがコンテンツの発信なら，サービスを構築して提供するのももちろんコンテンツの発信なのだが，この 2 種の発信には大きな隔たりがある。

サービスを利用する側はネットワーク接続環境とブラウザさえ用意すればよく，あとは創造力次第で独自のコンテンツを送り出すことができる。これに対してサービスを提供する側は，サーバを用意し運営するという資金面・労力面のコストがかかる上に，サービスを実際にプログラムとして実装するという技術力が要求されるのである。これらをクリアして，はじめて思いついた

[1] "Google Calendar" http://www.google.com/calendar/

[2] "Amazon.com" http://www.amazon.com/

[3] "twitter" http://www.twitter.com/

サービスをコンテンツとして発信できる。

このようなコストの格差が生じるのは当然のことなのだが，サービスを提供するための敷居が高すぎることは，ウェブサービスの進化を阻害しかねない。ウェブサービスのアイディアはまだまだ枯渇にはほど遠く，現に，上述したミニブログも字幕型動画投稿サイトもここ 2 年以内に登場した新種のサービスである。

アイディアを実現するための敷居を下げ，多くの人がウェブサービス提供に参加できる環境を作ることはオンラインコミュニケーションの世界をより豊かにしてくれるであろうと期待できる。

1.3 オンラインビジュアルプログラミング言語

前項で述べた問題点について，著者らはウェブサービスのアイディアを誰でも自由に具現化できる環境を実現することが解決になると考えた。

そして，この目標を満足するために，次の 2 つの機軸に沿った技術を開発することとした。

- **図形でプログラミングする言語**
  習得に時間がかからず，既存のソースも一目で読み取ることのできる，誰にでもわかりやすいプログラミング言語であること
- **ウェブ上で，ブラウザのみで利用できる開発環境**
  サーバの準備もファイルのアップロードも必要なく，ウェブ開発をゼロからすぐ始められること

この 2 軸に従い，オンラインビジュアルプログラミング言語たる「ゆば」開発に着手するに至った。

第2章　ビジュアルプログラミング

テキストによるソースコード打ち込みではなく図形的な描画によってプログラムを記述する言語を，ビジュアルプログラミング言語（以下，VPL と略す）と呼ぶ。

VPL の開発動機は多くの場合，本研究と同様に初心者にもわかりやすいプログラミングを実現することである。これは PC の利用が広い層にとって身近になったことと無関係ではない。そうした PC の普及には GUI（Graphical User Interface）が大きな役割を果たした一方で，GUI というプラットフォームによって VPL の実装は現実的なものとなった。GUI の普及は開発動機，実装の両面から VPL に関係深い。

こうした経緯から，VPL のターゲットとする利用者はプログラミング初心者，コンピュータ初心者であり，そこで必須となるのは「わかりやすさ」である。

- 記述ルールの理解しやすさ
- プログラム図形の読解のしやすさ

という二つの面からのわかりやすさが VPL にとっては存在意義ともいえる重要の要素となる。VPL をデザインするにあたっては, プログラミング言語としての整合性を実現するのはもちろん, この初心者にとっての二点のわかりやすさを損なうものにならないよう注意の払われる必要がある。

この視点から，VPL について概説する。

2.1　VPL における言語の定義

テキスト型言語において, 言語とは処理系によって受理されるキャラクタ列の集合を意味する。VPL にとってプログラムは図形構造である以上，受理すべきキャラクタ列というのは定義されない[1]。

文字列の集合として VPL を定義できないならば VPL の仕様とは何かと考えると，それは図形構造を構成する要素図形の集合と，図形同士の接続ルール，そしてそれらの意味論である。これらのルールに従って処理系が受理できる図形構造の集合が VPL にとっての言語であると言える。

テキスト型における言語をも包含して定義するならば，言語とは処理系の受理できるデータ構造の集合のことであるとも言える。以下の議論で言語という用語を使用した場合，この広義の言語のことを意味する。

2.2　言語パラダイム

VPL の言語仕様は，要素図形とそれらの接続で何を表現するかによっていくつかのパラダイムに分類される。

- データフロー型
- 制御フロー型

[1] もちろん多くの場合 VPL 開発環境は作品をディスク上にファイルとして保存する機能を持つから，ファイル出力のさいにキャラクタ列化するのは可能だとは言える．しかし，保存形式は VPL そのものの仕様ではなく環境依存の機能である．ファイル出力されたキャラクタ列を言語と考えるなら，保存形式が変われば別の言語になってしまうことになる．

• オブジェクト型

オブジェクト型は前二者とはやや別カテゴリに属する。データフロー型と制御フロー型は画面上に処理の内容を記述するためのものだが，オブジェクト型は画面上に処理の対象であるオブジェクトを配置するものである。この場合，各オブジェクトのメソッドを別に記述する必要があり，そのためにデータフロー型か制御フロー型の処理記述言語が必要である。実際には，処理記述言語として制御フロー型を採用することが多い。

2.2.1 制御フロー型言語

制御フロー型言語（以下，CFL: Control Flow Language と略する）は，アルゴリズムの実行順序を図形表現したものである。要素図形は命令で，それらを実行順に接続したものがプログラムとなる。

処理系はプログラムを先頭の命令からまず注目して実行し，接続されている次の命令へ注目を移して順に実行していく。命令間でのデータの受け渡しは，変数への代入と参照によって行われる。一般的に実用的なプログラムにおいて変数は複数必要であり，複数の変数は名前などの識別子を付加することで区別される。CFL は，図形だけでなく変数名まで読むことではじめて命令間のデータ授受関係が明らかになってプログラムの構造が理解できる言語体系である。

ロボット制御用の Scribbler[4], オブジェクト指向の VIPR[5]などがある。

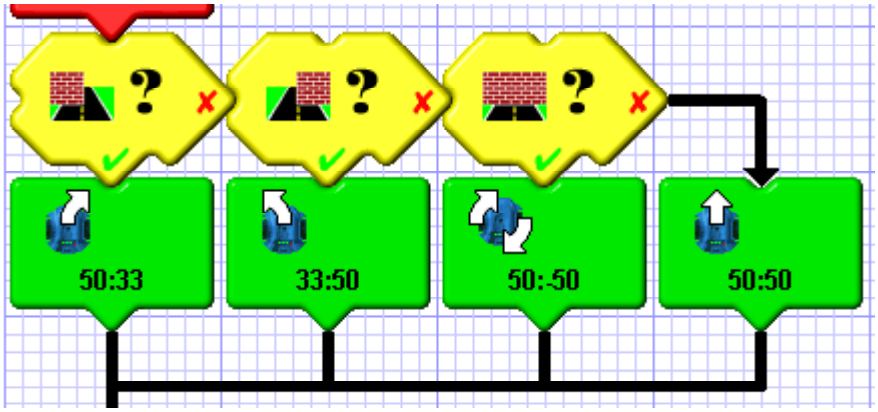

図 2.1 CFL のプログラム例（Scribbler）[4]より

2.2.2 データフロー型言語

データフロー型言語（以下，DFL: Data Flow Language と略称する）は，演算命令の間で受け渡しされるデータの授受関係を図形表現したものである。図形要素はそれぞれが演算命令である。演算命令にはその入力と出力である端子があり，命令同士の出力端子・入力端子を線でつないでプログラムとする。

DFL においてプログラマは，命令の実行順序を明示的には指定しない。順序を記述しないことは，並列的な処理を自然に記述することを可能にする。CFL では図形構造に加えて文字部分（変数名）を読まなければ処理は理解できないが，DFL において処理は図形構造だけで完全に表現されている。

実装例は多く，汎用の Show and Tell[6]や Marten[7], 科学計算用の DataVis[8], マルチメディア向けの Hyperflow[9], 信号処理用の LabVIEW[10], 音楽編集用の Pure Data[11]や MAX/MSP[12],

スクリーンセーバー設計用で MacOS 標準搭載の Quartz Composer[13]などが挙げられる。

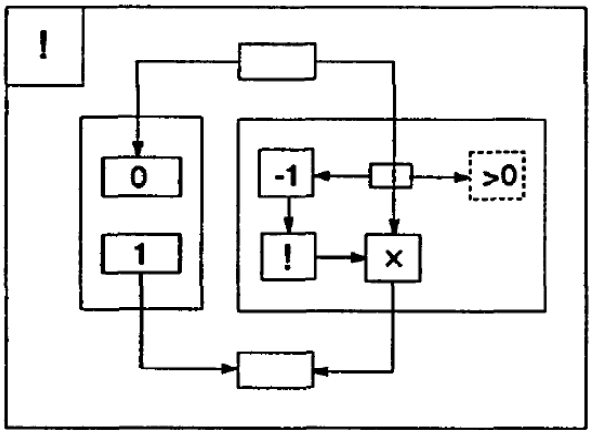

図 2.2 DFL のプログラム例（Show and Tell）[9]より

2.2.3　オブジェクト型言語

オブジェクト型言語（以下，VOL: Visual Object Language と略する）は，画面上に配置したオブジェクトに対しメソッドを記述してオブジェクトを操作する言語である。メソッドの処理を記述するために何らかのパラダイムの言語を必要とする。このメソッド記述は CFL でも DFL でも構わないが，オブジェクトの状態を変化させるのが処理の基本なので，参照透過性のある処理に向く DFL よりも変数書き換えを主体とした状態変化が基本の CFL と相性がよい。

メソッド記述言語には CFL と DFL 以外の選択肢として，パターンマッチ型がある。これは，オブジェクトの状態がパターン表のどれに該当するかパターンマッチし，オブジェクトが次にどのような状態になるかを該当するパターンごとに指定しておくという方式である。

VOL は画面上のグラフィカルなオブジェクトを動かすためのシステムなので，子供の教育用やゲーム作成用といった色合いが強くなる。

実装例としては，教育にも使える本格的オブジェクト指向開発環境として有名な Squeak[14]のほか，動く絵本を作る Viscuit[15]，プログラミング教育用の ToonTalk[16]，ゲーム作成用の Stagecast Creator[17]などがある。VOL の多くは子供を対象にした実装である。

第3章　言語デザイン

今回筆者らが開発を目指する「ゆば」は，ウェブシステムを誰でも手軽に構築できるようにすることを目的とする。言語仕様は，その目的に沿って設計していく。このとき重要な点は

- 記述ルールの理解しやすさ
- プログラム図形の読解のしやすさ

という VPL 共通の必須条件に加え，ウェブシステム特有の要求である

- データベース操作を簡潔に記述できる
- セッション管理を簡潔に記述できる
- HTML を意識させず簡単に画面出力を設計できる

を満たすことである。

3.1　データフロー型ビジュアル言語として設計する

VPL のパラダイムは CFL，DFL，OOL の 3 種類に大別される。これらの中からパラダイムを選定する。選定にあたり考慮すべき点として，ゆばに要求される動作には次のような特徴がある。

- グラフィクスは扱わない
- 対話的なプログラムは作成しない

グラフィクスを扱わないことにより，画面上のグラフィカルなオブジェクトを扱うための VOL は候補からはずれる。

対話的なプログラムとは，ユーザに必要に応じて入力を求め，それによって動作を変化させるプログラムのことである。ウェブシステム全体で提供されるユーザ体験は対話的だが，個々の HTTP リクエスト応答動作はリクエストに対して HTML 文書を返すだけであり，まったく対話的でない。HTTP リクエスト応答動作を記述するのがウェブプログラミングであり，対話性は求められていないことになる。

さて，CFL と DFL にはそれぞれ短所がある。

- CFL はどの演算がどの演算の結果を利用しているかというデータの流れを図形構造で表現できず，プログラムの読解しやすさで劣る。
- DFL は処理の順序を記述できず，どのタイミングで入力を求めるかという対話動作の記述がしにくい。

これらを比較したとき，対話性を求められないウェブプログラミングにおいては，DFL の短所はあまり問題とならない。対して，わかりやすさを追求するゆばの設計目標において CFL の欠点の読解しにくさは問題である。

よって，ゆばを DFL として設計することとした。

3.2　ゆば言語

前章で述べたように VPL において言語はプログラムを構成する図形の集合，図形の接続規則，および接続の意味論によって定義される。ゆばの言語仕様は次のように定義する。

3.2.1 構成図形

キャンバス

縦横のグリッドが入った白い矩形がプログラムごとにひとつだけ存在する。この矩形を**キャンバス**と呼ぶ。

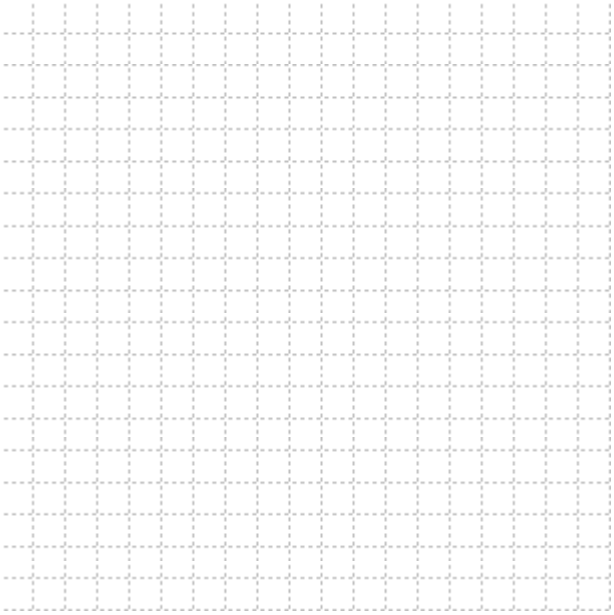

図 3.1 キャンバス

パネル

突起を持った矩形があり，これらは**パネル**と呼ぶ。突起をのぞく矩形部分は，特に**パネル本体**と呼ぶ。パネルには複数の種類があり，種類によって突起の数が違う。パネルの中には，突起の数を変更できるものと，突起の数が固定されているものがある。

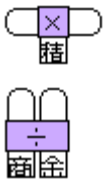

図 3.2 パネル

端子

パネルの突起には角の丸い突起と角のとがった突起の 2 種類があり，角の丸いものを**入力端子**，角のとがったものを**出力端子**と呼ぶ。両方をあわせて端子と呼ぶ。

パネルに付属していない端子もある。出力端子の形の独立した突起を**引数端子**と呼び，これは出力端子の一種である。入力端子の形の独立した突起は**結果端子**と呼び，これは入力端子の一種である（引数・結果端子はパネルに付属しない代わり，配置上の制約がある。次項で述べる）。

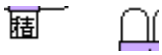

図 3.3 パネルの入力端子・出力端子

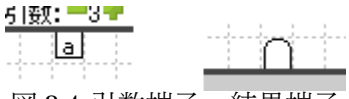

図 3.4 引数端子・結果端子

パイプ

様々な色の付いた，折れ目や枝分かれのある管状の図形があり，**パイプ**と呼ぶ。

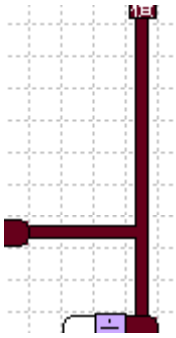

図 3.5 パイプ（中央の分枝状の茶色い図形）

3.2.2 図形の接続規則

パネルはキャンバス上に配置できる

パネルは種類にかかわらず，キャンバスに重ねて配置することができる。このとき，パネルはキャンバスのグリッドに沿うように設置され，グリッドが作るマス目をいくつか覆う形で重なる。パネルの端子は，ちょうどマス目 1 個を覆う大きさである。

キャンバスには，複数のパネルを配置できる。パネル同士が重なることはできない（同じマス目を複数のパネルが覆わない）。

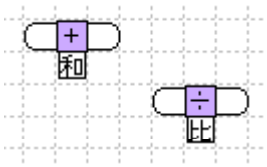

図 3.6 配置されたパネル

引数端子と結果端子はキャンバス上に配置できる

引数端子はキャンバスの上辺に沿い，マス目を各 1 個覆うように重ねて 0 個以上，結果端子はキャンバスの下辺に沿い，マス目を各 1 個覆うように重ねて 1 個以上配置できる。結果端子は 1 つ以上配置しないといけない。同じマス目を複数の端子やパネルが覆ってはいけない。

パイプは端子同士をつなぐように配置できる

パイプは枝分かれできるので，端が 2 つかそれ以上ある。パイプの端は必ず端子と重なっていないといけない。パイプと端子が重なっていることを，パイプと端子がつながっていると言う。パイプとつながっている端子のうち，出力端子は 1 つでないといけない。このとき，入力端子はどれも出力端子とつながっていると表現する。

パイプ同士が重なることはできるが，パイプの折れ目や分岐点が他のパイプと重なることはできない。またパイプは，パネルやつながっている端子以外の端子とは重なることができない。

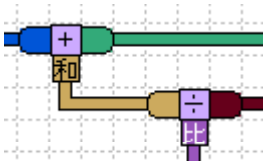

図 3.7 パイプでつながった端子

上記の制約を満たしていても，**循環参照**と呼ばれる禁止されたパイプの配置がある。任意のパネルに注目して，その入力端子とつながっている出力端子のあるパネルへと注目を移していくとき，元のパネルに注目が戻ることができるようなパイプ配置がそうである。

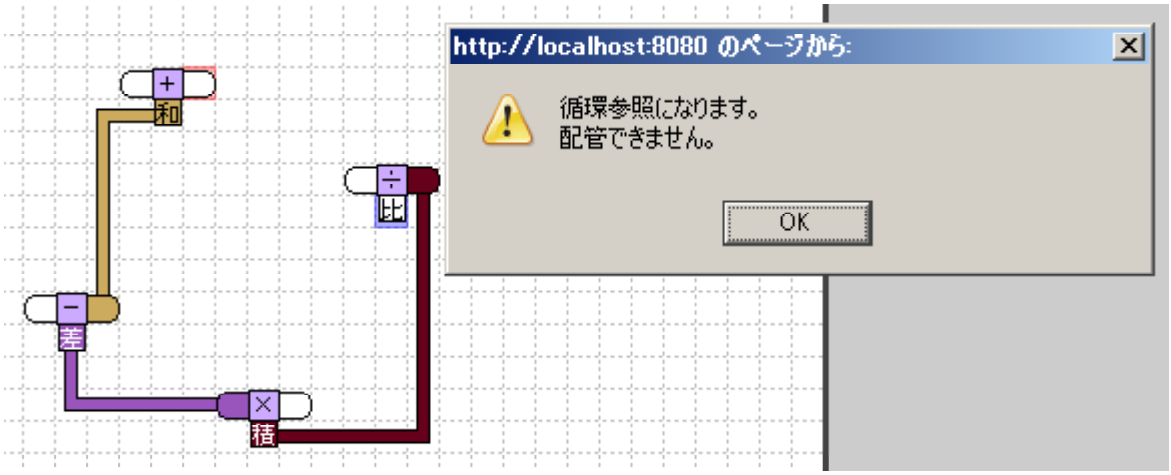

図 3.8 循環参照が起こるパイプ配置はできない

パネルのうち，端子にひとつもパイプがつながっていないものは，パネル本体が出力端子としてパイプとつながることができる。

3.2.3 意味論

出力端子へのデータの要求

出力端子はひとつだけデータを持つことができる。出力端子は一度データを持つと，そのデータはプログラムの実行機会の中では不変である。

出力端子には**データを要求**することができる。データを要求された出力端子は，データを持っていたならそのデータを返す。持っていなければ，所属するパネルに演算を要求する。演算が終わると出力端子はデータを持たされており，そのデータを返す。

パネル本体が出力端子となっている場合，データを要求されたパネルは，それ自身の種類をデータとして返す。

演算の要求

パネルには**演算を要求**することができる。パネルは演算を要求されると，パネルの種類ごとに定められた計算を行い，その結果を所属する出力端子に持たせる。演算の際，パネルは必要に応じてそれ自身の入力端子にデータを要求する。

入力端子が**エラー値**と呼ばれる特殊な種類のデータを返してきた場合，そのパネルは演算を中止し，すべての出力端子に同じエラー値を持たせる。エラー値は，パネルの演算の結果として発生することもあり，この場合もすべての出力端子がそのエラー値を持たされる。

入力端子へのデータの要求

入力端子にも**データを要求**することができる。入力端子はデータを要求されると，パイプでつながっている出力端子にデータを要求し，出力端子が返してきたデータを返す。入力端子がデータを要求されたのにパイプがつながっていなければ，エラー値を返す。

プログラムの実行

プログラムには，定められた個数の順序づけされたデータを与えて，**実行**させることができる。与えるべきデータの個数は，プログラムに存在する引数端子の数と同じである。引数端子と与えるデータは，引数端子の左端からの順番とデータの順序によってそれぞれ対応する。

プログラムが実行されると，結果端子がそれぞれ，自分につながるパイプにつながる出力端子にデータを要求していく。要求は順に行われる。すなわち，ある結果端子のデータ要求と受け取りが終了すると，次の結果端子がデータ要求をする。要求をする順番は，キャンバスの下辺に並んでいる結果端子群の，左側からである。

プログラムの実行開始がきっかけでデータ要求を出すものには，結果端子の他に，特殊な種類のパネルの入力端子がある。記憶スロット保存パネルと呼ばれる種類のパネルがそのパネルである（4.8.5 を参照）。

すべての結果端子がデータを受け取ったらプログラムが終了し，結果端子が受け取ったデータがプログラムの実行結果として得られる。結果端子が受け取ったデータは，左の結果端子のものから順に順序づけされて，プログラムの実行結果として出力される。

3.3 型体系

ゆばが DFL を採用したのは，図形表現だけで処理を読解できるようにするという目的があることは 3.1 で述べた。ここで，データ伝達をパイプという図形で表現しようとすると，そのデータの型も図形構造の中で表現できるかという問題が生じる。

データ型には無数の種類がある。整数，文字列などといった基本型のほか，基本型のデータを組み合わせた複合型（配列や連想配列，構造体，関数型）があり得るためである。

パイプを表現する図形の画素数は有限であり，無限の種類の型を図形で表現しきることはできない。データの型を図形として可視化できないということになる。

ここで，もしゆばの型システムに静的な型付けを採用するなら，型の合わない端子同士をつなぐパイプは置けないというルールが必要になる。しかし，画面上の図形構造からは見えないデータ型によって接続が制限されるという仕様は，初心者プログラマにとって不可解な動作であり，わかりやすさという開発目的に反している。

そこで，データはすべて動的に型付けされるものとした。そして，必要に応じて内部で自動的に変換が行われるようにし，パイプの接続に関して型による制約は設けない。

このような動的データ型体系をプロジェクトの一環として開発し，**バリアント型(Var)**と名付けた。バリアント型の詳細については 4.6 節および参考文献[18]を参照のこと。

3.4 画面設計

Web ページを出力することは，HTML 文書を出力することである。

しかし，HTML 文書を生成する作業をプログラマにさせるのでは，手軽さという開発目的からはずれてしまう。

そこで, 画面表示の生成はパネルとパイプによるプログラミングと別枠でできるようにしたい。

ゆばでは，画面に表示させたいテキストのみをプログラマに打ち込ませることにした。テキスト中に簡易タグを埋め込むことで HTML の装飾機能を利用することができる。

テキストにはまた，「差し替え」を意味するタグを埋め込めるようにする。差し替えタグを含んだテキストには，差し替えタグに対応した個数のデータを与えることで，タグのある箇所をそれに差し替えたテキストデータを受け取ることができる。

データを与えることでデータを出力する動作は，プログラムの動作と同じである。そのため，テキストはプログラムと同様に扱われる。プログラムとテキストを総称して**モジュール**と呼ぶ。

3.5 部品化

モジュールは，そのまま実行するだけでなく，プログラムの部品としても利用できる。部品としてモジュールを利用する場合は，そのモジュールはパネルとなる。そのパネルが持つ入力端子と出力端子は，そのモジュールがプログラムであれば引数端子と結果端子に対応するだけある。そのモジュールがテキストであれば，パネルの入力端子は差し替えタグに対応するだけあり，出力端子はひとつだけあって完成テキスト出力に対応する。

プログラムは，そのプログラム自身をもパネルとして利用し，再帰呼び出しを記述することができる。

3.6 記憶

実用的な Web サービスを作るにあたっては，プログラムのある実行機会に発生したデータが，必要に応じて別の実行機会にも利用できるようになっていないといけない。こうしたデータの長期記憶機能を，一般的なウェブシステムは DB（DataBase）操作によって実現している。

DB 操作を記述するのは，SQL クエリを記述しないといけないなど煩雑な過程である。プログラマに煩雑な DB 操作を記述させるのはゆばの設計目的とあわない。手軽さとわかりやすさのためには，あくまで図形操作だけで簡単に長期記憶機能を記述できるようにしたいのだ。この要求を実現するために，記憶スロットという機能を用意する。

記憶スロット機能は，データを永続的に保管する入れものを複数用意できるようにし，それぞれの入れものについて格納と取り出しをするパネルを用意するというものだ。この機能の詳細については次章 4.7 で述べる。

第4章　仕様

前章で述べた設計方針を元に，実際にどのような動作仕様，インタフェースを持った言語処理系を開発してきたかを述べる。

4.1 利用モデル

4.1.1 利用者

ゆばの利用者は 2 種類に分けられる。ゆばを用いてウェブシステムを作成し公開する**クリエイタ**と，クリエイタの作ったシステムを利用する**ユーザ**である。

クリエイタになるには，アカウント名とパスワードを決めてアカウントを取得すればよい。

4.1.2 環境

クリエイタが開発環境を利用するには，Javascript の使用できるブラウザ[1]を用いる。プラグイン等，特に準備すべきソフトウェアはない。

作成されたウェブシステムをユーザが利用するのにもブラウザを用いるが，このブラウザは Javascript 対応である必要はない。

4.2 プロジェクト

プロジェクトは，複数のモジュールを収める入れものである。モジュールは必ずプロジェクトのひとつに属する。通常，クリエイタの作品であるウェブシステムは 1 つのプロジェクトで実現

[1]動作確認は Microsoft Internet Explorer 6.0 および Mozilla Firefox 2.0 のみで行っている。

されている。

クリエイタは複数のプロジェクトを作成することができる。プロジェクトはそれを作成したクリエイタに属し，そのクリエイタだけが変更を加えられる。

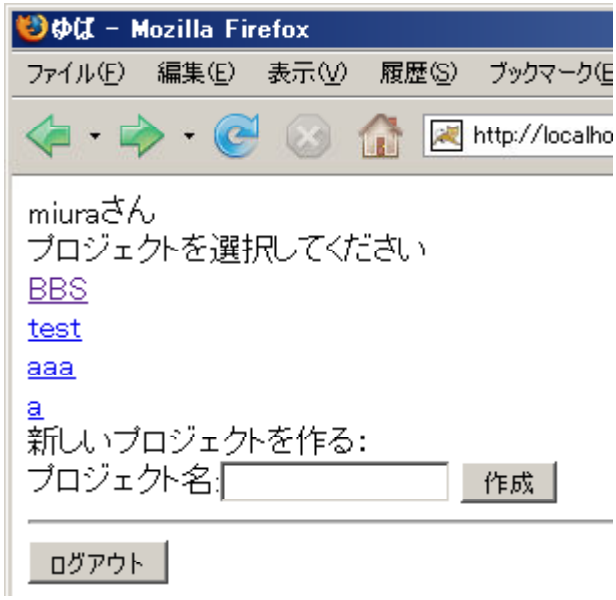

図 4.1 クリエイタ用のプロジェクト一覧画面

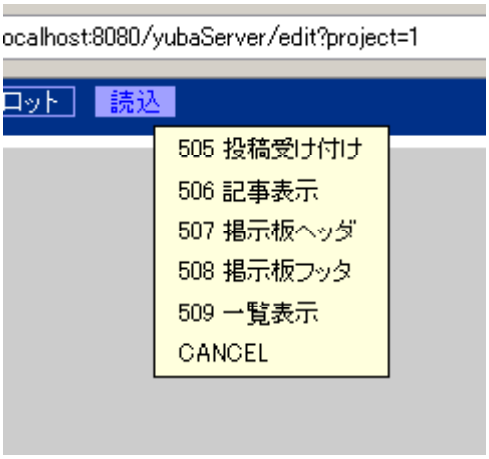

図 4.2 プロジェクトのモジュール一覧

4.2.1 プロジェクトの作成

プロジェクトは名前を持つ。同一のクリエイタに属する名前の同じプロジェクトは作れない。作成したプロジェクトをウェブシステムとして公開したときは，プロジェクトの名前がウェブシステムの名前となる。

プロジェクトを新規作成するときは，クリエイタ各自に用意されたプロジェクト一覧ページで，名前を指定して新規作成ボタンをクリックする。作成したばかりのプロジェクトはモジュールを含まない空のプロジェクトである。

4.2.2　プロジェクト開発環境

プロジェクトの開発に入ると，ブラウザに図 4.3 の画面が表示される。この画面で，プログラムやテキストを編集したり，プロジェクトの設定を変更したりする。

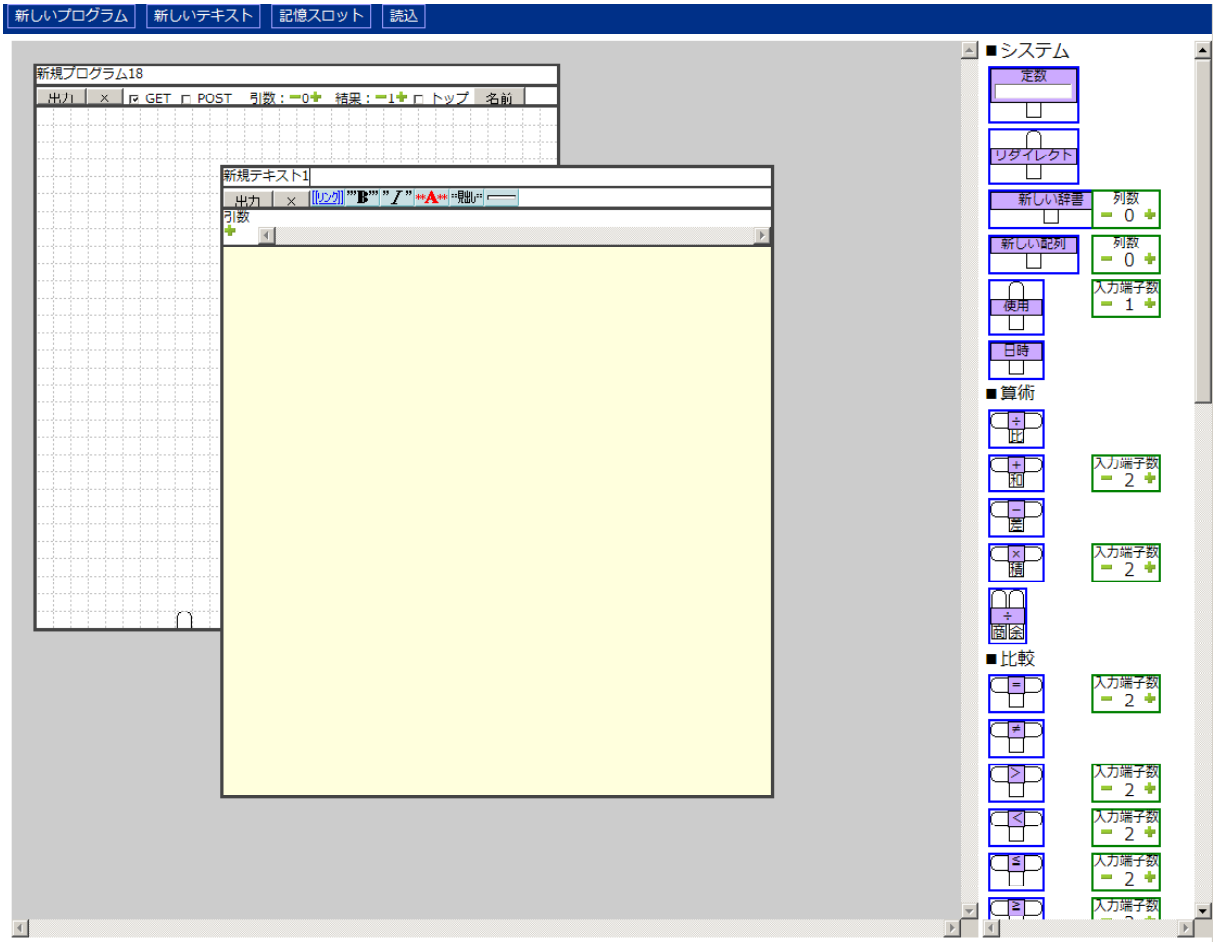

図 4.3 プロジェクト開発環境

作業領域

画面左側の灰色の領域は作業領域である。この領域に，プログラムやテキストの編集ウィンドウを表示して編集作業を行う。編集ウィンドウは複数表示することができる。プログラムの編集ウィンドウはキャンバスフレーム，テキストの編集ウィンドウは**エディタ**と呼ぶ。キャンバスフレームはキャンバスを 1 枚含んでいる。

キャンバスフレームとエディタの他，プロジェクトに属しているモジュールの一覧を表示するウィンドウもこの作業領域に表示される。4.7 で述べる記憶スロットの一覧を表示するウィンドウの表示も同じくここに表示される。

部品領域

画面右側の白く縦長の領域は部品領域である。ここにはいくつかのカテゴリに分類されたパネ

ルの見本が縦に並んでいる。利用できるパネルの種類すべてについて見本があり，これらを**スタンプ**と呼ぶ。スタンプをクリックしてからキャンバス上でクリックすることで，キャンバスにスタンプと同じ種類のパネルを配置できる。パネルの種類は 4.8 で後述するが，プロジェクトに属するモジュールのスタンプのほか，システムで用意している**標準パネル**のスタンプも部品領域に置いてある。

一部のパネルは，入力端子の数を自由に増減できるようになっている。例えば標準パネルの「＋（加算）」は入力端子の数を増やすことで，任意の項数の加算をパネル 1 枚で表現できる。このように入力端子の数を変更できる種類のパネルは，スタンプの右側に増加ボタン（＋）と減少ボタン（－）が並んでいる。端子の数を変更できる種類のパネルでも，キャンバス上に配置されたものの端子数は変更できない。

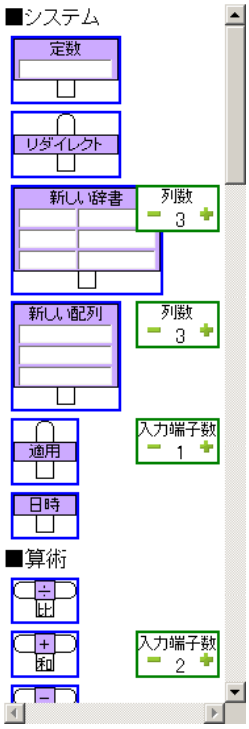

図 4.4 部品領域

4.2.3 プロジェクトの公開

プログラムは他のモジュールをパネルとして利用できると前章で述べたが，通常利用できるのは同じプロジェクトに所属するモジュールだけである。

しかし，汎用性の高いモジュールを作ったときなど，そのモジュールを他のプロジェクトからも呼び出して利用したいことがあるかもしれない。そういったときのために，モジュールを他プロジェクトへ公開するよう設定することもできる。

利用する側のプロジェクトでは，公開されているモジュールの属するプロジェクトを取り込むよう宣言することで，公開モジュールをパネルとして利用できるようになる。取り込み宣言をすることを**インポートする**という。

4.3 モジュール

モジュールはプログラムとページの総称である。モジュールには 2 種類の働きがある。固有の URL を持って HTTP アクセスを受け付けることと，すなわちウェブページを表示することと，パネルとしてキャンバス上に配置されることである。

4.3.1 ウェブページの表示

モジュール固有の URL，システムが内部的にモジュールに振った ID 番号によって「http://システムのドメイン名/yuba/execute/モジュール ID」という形式で決定される[1]。

この URL に HTTP リクエストが来ると，モジュールが実行される。その際モジュールに渡されるデータは，HTTP リクエストに含まれるパラメータである。HTTP リクエストによるパラメータの渡し方には GET 方式と POST 方式があるが，どちらの方式で値を受け取るかはモジュールごとに設定しておく必要がある。それは編集ウィンドウ上に設定ボタンがある。GET/POST どちらでも受け取るという設定も可能である。

モジュールの実行が終了すると，出力が 1 個だけのプログラムやテキストならその出力結果が，出力の複数あるプログラムなら最左端の端子の出力結果が，HTTP レスポンスとしてブラウザに出力される。

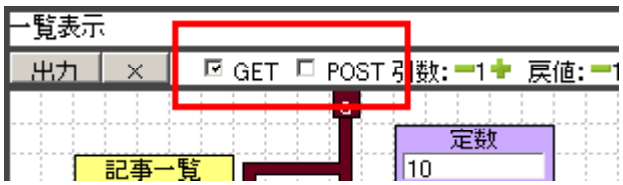

図 4.5 GET/POST メソッド設定ボタン

4.3.2 モジュールのパネル化

モジュールは，保存してプロジェクトに登録されればパネルとして使うことができるようになる。パネルの形状は，入力・出力端子の数から自動的に成形される。

4.4 プログラム

プログラムはモジュールの一種で，キャンバス上にパネルとパイプを配置して計算処理を記述したものである。

[1] 通常ユーザもクリエイタもこの URL を意識する必要はない．ページ同士でリンクを張るときも，後述する装飾タグ記法によって，モジュール名を書くだけでリンクが述できるからである．

4.4.1　プログラムの新規作成と呼び出し

プロジェクトに新しくプログラムを追加するときには画面上段の「新しいプログラム」ボタンをクリックする。すでにあるプログラムを読み込むときには画面上段の「パネル一覧」をクリックして出てきた一覧ウィンドウの中から編集したいプログラムを選ぶ。

新規作成するか既存のプログラムを読み込むと，作業領域に新しいキャンバスフレームが開く。

4.4.2　キャンバス

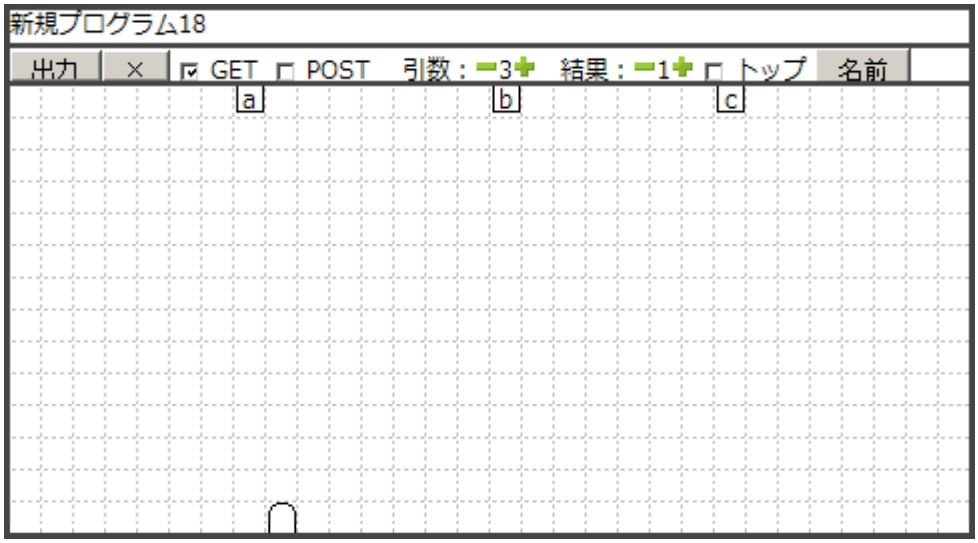

図 4.6 キャンバス。3 入力 1 出力のプログラム

キャンバスの上辺には引数端子が，下辺には結果端子が並ぶ。引数端子と結果端子は，そのプログラムの入力と出力に対応している。引数・結果端子の数はキャンバスフレームの端子数増減ボタンをクリックすることで変更できる。

端子を増やすときは，既存の端子の右側に新しい端子が追加されていく。端子には自動で a から順にアルファベットが名前として振られていく（z の次は aa, ab, ac, …となる）。意味を持った名前をつけたいときは，端子を右クリックして命名ダイアログを呼び出せばよい。

新しい端子が増えても端子が右側に詰まって配置されないよう，既存の端子の位置も自動的に調節されて均等に並べられる。

端子を減らすボタンを押すと，右側の端子から順に削除されていく。ただし，すでにパイプで配管されている端子を削除することはできない。最右端の端子がすでに配管済みの場合は削除ができない。先に，削除したい端子につながるパイプを削除しないといけない。

端子を減らしたときは，右側に空白ができないよう，残った端子の位置が自動的に調節される。

図 4.7 加算パネルのスタンプと端子数増減ボタン

4.4.3　キャンバスへのパネルの配置

キャンバス上で計算処理を記述するには，演算子または関数にあたるパネルをまず配置する。配置するには，配置したい種類のパネルのスタンプを右側の部品領域から探して，マウスクリックで選択する。次にマウスをキャンバス上に持って行くと，スタンプの形状に合わせてマーカが出る。他のパネルと重なるなどで配置できない場合は，マーカが灰色になって配置できないことを知らせる。配置したい位置にマーカを合わせてクリックすると，そこにパネルが配置される。

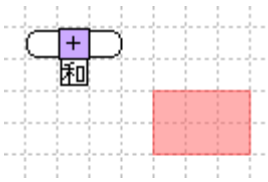

図 4.8 パネルが配置される位置にマーカが点灯

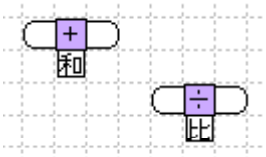

図 4.9 クリックによってパネルが配置された

4.4.4　パイプの配管

入力端子がどの出力端子にデータを要求するのかを決めるパイプは，出力端子と入力端子をひとつづつマウスでクリックすることで配置できる。パイプの経路は自動的に決定される。図形が混み合っている場合には，経路が決定できずパイプが配置できないこともある。そのような場合にはフレームを拡大してマス目を増やさないといけない。循環参照を起こすパイプも配置することができない。パイプを配置することは，**配管する**とも言う。

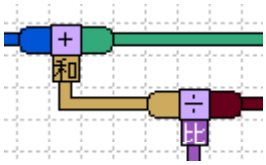

図 4.10 配管

パイプは，出力端子と入力端子の間だけでなく，パネル本体と入力端子の間にも配管することができる。これは，パネルそのものを一級データとして出力するための記法である。パネルそのものをデータとして扱うことについては 4.6.3 で後述する。

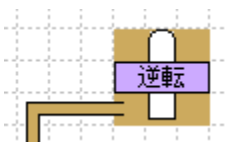

図 4.11 パネル本体を出力端子として扱う

4.4.5　オブジェクトの削除

パネル，端子やパイプを総称して**オブジェクト**と呼ぶ。一度配置したオブジェクトはキャンバスから削除することができる。削除したいオブジェクト上で右クリックし，出てくるコンテキストメニューから削除を選択すればよい。

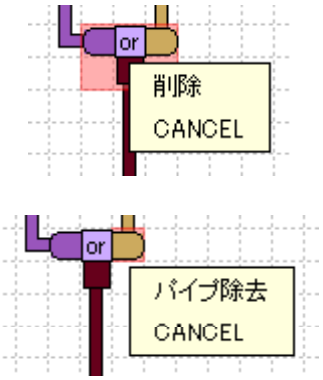

図 4.12 パネル本体と入力端子のコンテキストメニュー

4.4.6　設定

プログラムは，名前のほかに GET・POST どちらのメソッドで値を受け取るかを設定する必要がある。設定ボタンはキャンバスフレーム上部のツールバーにまとめられている。

名前を設定する際に使用できない文字は，[ ] | ? の 4 種である。これらが使用できないのは，後述するページ記法の特殊文字と衝突するためである。文字の使用制限は，テキストを命名する際も同様である。

図 4.13 プログラム設定ツールバー

4.4.7　サイズの変更

キャンバスの大きさは，そこに入るマス目の数を規定するのでプログラムにとって重要な要素である。例えば，キャンバスの横幅以上の入口端子や出口端子を置くことはできない。横幅一杯まで端子を増やしても足りないときには，キャンバス自体を広げる必要がある。複雑な処理を記述したいのにキャンバスが狭くてパネルが配置しきれない場合もありうる。

キャンバスの大きさは，キャンバスの端の黒い枠をマウスでドラッグすることで変更できる。開発環境の作業領域はスクロールできるので，作業領域より大きなキャンバスを作っても編集作業は可能である。

4.4.8　プログラムの保存

プログラムはツールバーの保存ボタンをクリックするか，キャンバスを閉じることでサーバに保存される（キャンバスを閉じるとき破棄終了することもできる）。

再帰的呼び出しをするプログラムを一から記述するには，新規作成したプログラムの引数・結果端子の数を調整してから，いったん保存する必要がある。

4.5　テキスト

テキストはモジュールの一種で差し替えタグを持つ文字列である。これを実行すると，差し替えタグ部分を与えられたデータで置換した文字列を出力する。

テキストの出力は最終的に HTTP レスポンスとしてブラウザに出力されるが，テキストを記述するのに HTML マークアップは不要である。WikiWikiWeb に似た簡易タグでマークアップするようになっている。その簡易タグもわざわざ暗記する必要はなく，入力支援ボタンによりワンクリックで入力できるようになっている。

4.5.1　テキストの新規作成と呼び出し

プロジェクトに新規にテキストを追加するときには画面上段の「新しいテキスト」ボタンをクリックする。すでに作ってあるテキストを編集する場合には，モジュール一覧ウィンドウを呼び出して，一覧の中から編集したいテキストを呼び出す。

4.5.2　エディタ

テキストの編集ウィンドウは，エディタと呼んでいるとおり，一般的なテキストエディタである。そして，基本的なエディタとしての編集機能に加え，マークアップ支援機能を持っている。マークアップ支援は，アイコン表示されたタグ入力ボタンをクリックすることでタグが文章中に挿入されるというものである。

図 4.14 エディタ

テキストに使うマークアップタグは，その性質によって 2 種類の分けられる。ひとつは，ブラウザで表示されたときの見た目を装飾するためのテキストである。もうひとつは，差し替えの対象となるタグである。エディタのツールバーは 2 段になっており，装飾タグの入力ボタンの列と差し替えタグの入力ボタンの列にわかれている。

4.5.3 装飾タグ

テキスト中に書き込まれた装飾タグは，HTTP レスポンスの際に HTML タグに変換される。装飾タグには次のような種類がある。記法と，それにより生成される HTML マークアップを示す。

- 強調（シングルクオーテーション 3 つで囲む）

  ```
  '''強調したい文字列'''
  ```

  強調したい文字列

- 斜体（シングルクオーテーション 2 つで囲む）

  ```
  ''斜体にしたい文字列'''
  ```

  斜体にしたい文字列

- 赤文字（アスタリスク 2 つで囲む。アスタリスクで挟んで色指定もできる）

  ```
  **赤くしたい文字**
  ```

  赤くしたい文字列

  ```
  **blue*青くしたい文字**
  ```

  青くしたい文字列

- リンク（大括弧で二重に囲む）

  ```
  [[リンク先パネルの名前]]
  ```

  リンク先パネルの名前

- 表示を指定したリンク（大括弧で二重に囲んだ中を縦棒で区切って右側に文字列を指定）

  ```
  [[リンク先パネルの名前|表示する文字列]]
  ```

  表示する文字列

- 横棒（一行に，ハイフンを 4 つだけ）

  ```
  ----
  ```

  <hr/>

- 見出し行（行の先頭と末尾に等号を付ける。等号を重ねることで見出しレベルを下げられる）

  ```
  =大見出し=
  ```

  <h1>大見出し</h1>

  ```
  ==中見出し==
  ```

  <h2>中見出し</h2>

  ```
  ===小見出し===
  ```

```
<h3>子見出し</h3>
```

- 箇条書き（行頭にアスタリスクを付ける。アスタリスクを重ねることで入れ子を深くできる）

```
*項目
**子項目
```

```
<ul>
  <li>項目
    <ul>
      <li>子項目</li>
    </ul>
  </li>
</ul>
```

- 連番付き箇条書き（行頭に井桁を付ける。井桁を重ねることで入れ子を深くできる。箇条書きと連番付き箇条書きはお互いに入れ子になれる）

```
#項目
##子項目
```

```
<ol>
  <li>項目
    <ol>
      <li>子項目</li>
    </ul>
  </li>
</ul>
```

- フォーム（ハイフンを 2 つで v を囲むとフォーム先頭，^を囲むとフォーム末尾。フォーム先頭にはリンクタグを並べて書き，投稿先を指定する。コロンを挟んでメソッドも指定できる）

```
--v--[[投稿先]]:POST
--^--
```

```
<form action="execute/パネル ID" method="POST">
</form>
```

- 入力欄（疑問符と大括弧で囲む。縦棒で区切ってデフォルト値も指定可能。疑問符で区切ってサイズの指定もできる）

```
[?項目名?]
```

```
<input type="text" name="項目名" value=""/>
```

```
[?項目名?20?]
```

```
<input type="text" name="項目名" size=20" value=""/>
```

```
[?項目名?80?5]
```

```
<textarea name="項目名" rows="5" cols="80"><textarea>
```

- パスワード入力欄（疑問符 2 つと大括弧で囲む。縦棒で区切ってデフォルト値も指定可能）

[??項目名??]

<input type="password" name="項目名" value=""/>

- 非表示フォームデータ（疑問符 3 つと大括弧で囲む。縦棒で区切って初期値も指定可能）

[???項目名|値???]

<input type="hidden" name="項目名" value="値"/>

- 投稿ボタン（感嘆符と大括弧で囲む）

[!送信!]

<input type="submit" value="送信"/>

- 装飾の抑制（中括弧で二重に囲む）

{{この中の記述は'''装飾'''されない}}

この中の記述は'''装飾'''されない

前述したパネル名の命名制限（ [ ] | ? の 4 文字が使用できない）はこれらのタグとの衝突を防ぐために設定されたものである。なお，" < > & など HTML 記法と衝突しうる文字は，変換の際に自動的にエスケープされる[1]ため使用が許されている。

装飾タグは，すべてツールバーのボタンから入力できる。囲んで使うタグも，文字列を範囲選択した上でボタンクリックすることで，選択されていた文字列を囲む形で入力される。

4.5.4 差し替えタグ

テキスト中で差し替えタグを使うには，名前を付けた**差し替え項目**を用意する必要がある。a という名前の差し替え項目を用意すると，[%a%]という差し替えタグが使えるようになる。こうすると，テキストを実行したとき，差し替え項目 a に対応した入力データで[%a%]が置換される。

プログラムの引数端子が順に並んでいて順番を持っているように，差し替え項目も順番を持つ。差し替え項目の場合は，作成された順が順番である。

差し替え項目を追加するには，エディタのツールバーにある差し替え項目追加ボタンをクリックする。項目名を設定すると，ツールバーには差し替えタグを入力するためのボタンが出現する。

項目名には，タグの記法と衝突する % 以外の文字を使用することができる。装飾タグとして解釈可能な文字列を項目名に使うことは，問題ない。差し替えは装飾タグの解釈に先立って行われるから，誤って装飾タグと解釈される前に置換されてしまうのである。

差し替え項目ボタンは入力支援ツールであると同時に，このページが持っている差し替え項目の一覧としての機能もある。差し替え項目を削除するときは，ボタンの右隣の削除ボタンをクリックする。

[1] それぞれ", <, >, & という表記に変換される。この記法を文字実体参照と呼ぶ。

図 4.15 差し替えタグを埋め込んだテキスト

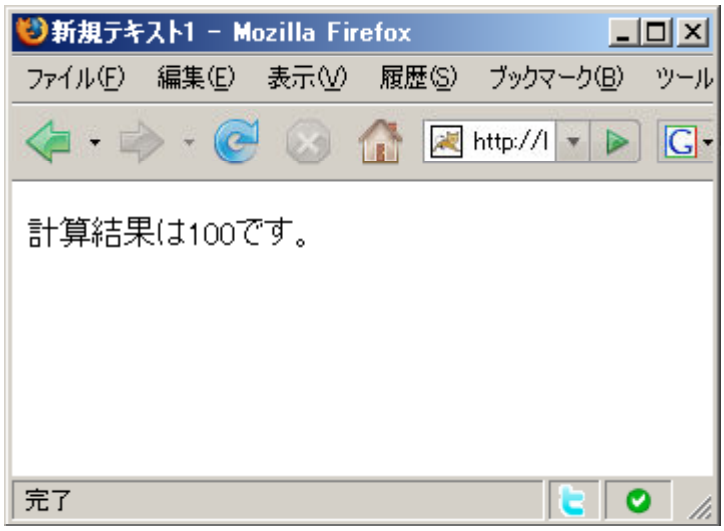

図 4.16 差し込み結果

4.6 データ・データ型

端子が持ち，パイプを通じてパネル間でやりとりされるデータは「生成されたら変化しない」という原則がある。たとえば配列データには新しく要素を格納するという操作ができない。既存の配列に要素を追加した新しい配列を作るという操作をすることになる。

データには以下の種類の型がある。それぞれの型は必要に応じて自動的に型変換される。

4.6.1 プリミティブ型

単独の値からなる基本的なデータ型である。以下のような型の集合である。

整数型

数値型の一種で，32 ビットの符号付き整数である。整数との四則演算を実行すると整数の結果を返す。実数との四則演算では実数を返す。それ以外の型との四則演算では，相手を数値型に

変換した上で演算結果を返す。
論理演算の引数になったときには，ゼロ値ならば false に，それ以外なら true に変換される。

実数型

数値型の一種で，64 ビットの浮動小数点実数である。どの型と四則演算を実行しても実数の結果を返す。
論理演算の引数になったときには整数型と同様に変換される。

真偽値型

真か偽かの 2 値をとる。数値型に変換されるときには整数型の 1 または 0 になる。

文字列型

可変長の文字列である。数値演算の引数となったときには自動で数値型に変換される。数値型に変換されるときは，数値として解釈できないなら 0 になる。数値として解釈できるなら，小数点を含まず 32 ビット整数の範囲に収まるときは整数型に，そうでなければ実数型になる。
論理値型に変換されるときには，

- 文字列が数値として解釈でき，ゼロでないなら true に，ゼロなら false に
- そうでなく，文字列が（大文字小文字を問わず）「false」，「no」，「偽」，「いいえ」，空文字列ならば false
- 上記どちらでもなければ，true

のルールで変換される。

ナル型

無効な値を意味する型で，データは持たない。ナル型の値はナル値と呼ぶ。配列の無効な番号の要素を取り出そうとした場合などにこの型の値が返る。数値型に変換するとゼロに，真偽値型に変換すると false に，文字列型に変換すると空文字列になる。

エラー型

パネルの演算結果がエラーであったことを示す型で，データとしてエラーメッセージを持つ。エラー値を入力に取ったパネルの演算結果もエラーになる。

4.6.2 コレクション型

エラー型以外の任意の型のデータを含むことができる型である。コレクション型には格納，取り出し，要素数を求める，キーの一覧取得，要素の一覧取得という特有の操作がある。プリミティブ型のデータに対してこれらコレクション的操作をした場合に，そのデータは「要素にそのデータのみを持つ長さ 1 の配列」として扱われる。
逆に，四則演算などプリミティブ型を対象とした操作をコレクションに対して行った場合は，型によって動作が異なる。

配列型

要素を任意の個数，順番をつけて格納するコレクションである。格納された要素には 0 から始まる連続した要素番号が振られる。

現在の最大の要素番号を超える要素番号に値を格納しようとした場合，要素番号が連続するよう，格納する要素番号までの間の要素番号にはダミーの要素としてナル値が格納される。

要素番号として実数や文字列が与えられた場合は，整数に変換されて解釈される。配列から値を取り出すとき，格納されていない要素番号が指定された場合は，ナル値を返す。

キーの一覧取得に対しては，0 から要素数-1 までの連続した整数の配列を，要素の一覧取得に対しては自分自身を返す。

プリミティブ型を対象とした操作を配列型に行うと，0 番要素に対してその操作をした結果が返る。

連想配列型

文字列をキーにして，要素に順序をつけず格納するコレクションである。すでに存在するキーで要素を格納した場合には要素が差し替えられ，まだ存在しないキーで格納した場合には新規格納になる。

存在しないキーで要素を取り出そうとすると，ナル値を返す。

キーの一覧取得と要素の一覧取得をすると，それぞれの一覧が配列型で返る。順序は不定である。

プリミティブ的操作に対しては，空文字列をキーとする要素（それがなければナル値）に対してその操作をした結果を返す。

4.6.3 その他の特殊な型

パネル型

出力端子ではなくパネル本体から配管されたパイプに流れるデータはパネルの種類であり，このデータの型がパネル型である。パネル型データは後述する適用パネルを使うことで，入力を与えて適用することができる。

パネル型以外の型のデータに適用操作を行おうとすると，0 入力 1 出力でそのデータそのものを出力するというパネル型データに変換されて適用が行われる。

パネルに四則演算や配列操作など，一般データ型的操作を行おうとすると，パネル型データはナル値と同様に振る舞う。

リダイレクト指令型

HTTP レスポンスには，HTML 文書による応答だけでなく，別の URL へリクエストし直せという指令もある。このリダイレクト指令にあたるデータ型としてリダイレクト指令型が用意されている。

転送指令型は最終的な HTTP レスポンスとして出力されるためだけの型であり，演算の対象で

はない。ほとんどの演算操作に対して，エラー値として振る舞う[1]。

4.7 記憶スロットシステム

ファイル入出力やデータベース操作に代わるデータ保存機能をゆばで提供するのが記憶スロットシステムである。

このシステムはデータ保存機能だけでなく，セッション管理機能も提供する。セッション管理機能とは，同じブラウザから複数回 HTTP アクセスを受けたとき，その複数回の応答動作の中で共通したデータコンテナにアクセスできるようにする仕組みである。セッション機能を利用することで，個別のユーザを認識してのサービスや，状態遷移のあるサービスを提供できるようになる。

4.7.1 記憶スロット

このシステムを利用するには，まずプロジェクトに**記憶スロット**を用意しなければならない。記憶スロットは，プロジェクトに属するプログラムが共通して使えるデータコンテナである。プロジェクトは複数のスロットを持つことができ，それぞれのスロットは名前で区別する。

画面上段の「記憶スロット」ボタンをクリックして，記憶スロット一覧画面を呼び出し，新規作成を選ぶことで作成できる。

記憶スロットには全体記憶と個別記憶と呼ぶ 2 種類がある。それらについて述べる。

全体記憶スロット

全体記憶スロット，そのスロット固有の単一のデータコンテナを持ち，全体記憶スロットへのアクセスはその固有コンテナへの読み書きとなる。

使用例としては，ウェブ掲示板システムを構築するとき，全体記憶スロットに記事一覧データを格納するなどがある。

個別記憶スロット

個別記憶スロットは，HTTP リクエストしてきたクライアントの個体を区別した上で，クライアントごとに独立したデータコンテナを用意する。アクセス元のクライアントによって，個別記憶スロットにアクセスしたとき読み書きすることになるコンテナは異なる。

HTTP クライアントの個体認識は通常 Cookie を利用し，Cookie の利用できないクライアントに対してはリンク URL に ID 文字列を埋め込むことで実現している。

個別記憶スロットを使用すると，例えばウェブ掲示板システムを構築するとき，記事を投稿してきた Web クライアントごとに投稿者名を記憶しておき，次回記事編集画面を出したときに投稿者名欄を最初から埋めておくといったことができる。

[1] コレクションへの格納と後述する記憶スロットへの格納ができる点だけがエラー値とは異なる.

4.7.2　記憶スロットの読み書き

記憶スロットを扱うプログラムにおいて，記憶スロット内のデータは特別なパネルからの出力として読み出せる。記憶スロットへのデータの格納は，やはり特別なパネルに入力しておくことで，プログラム終了後に実行される。

プログラムの扱う記憶スロットは，実行が開始されてから終了するまでの間，自動的にロックされる。

4.8　プログラム部品

プログラムを作るのは，キャンバス上にパネルとパイプを配置する作業である。キャンバス上で部品として使えるパネルには多くの種類があり，それらのスタンプは開発環境の部品領域に分類されて置かれている。

多くの種類があるスタンプの働きと用法を，クリエイタが名前と形状だけをみて理解するのは難しい。スタンプにマウスを近づけると解説文がポップアップするようになっている。

この節では，標準パネルの種類と働きを述べる。

4.8.1　定数パネル

プログラム内に定数値を置くためのパネルが定数パネルで，0 入力 1 出力のパネルである。定数パネルは配置した時点では空文字列を表しており，配置後に入力欄をクリックすることで，任意の文字列を入力できる。

入力できるのは文字列だけで，数値や真偽値などを打ち込みたい場合も「123」「いいえ」などと文字列で入力する。数値や真偽値として扱われないといけないときも，そのときに自動的に変換されるので問題ない。

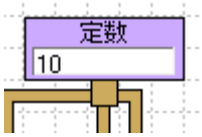

図 4.17 定数パネル

定数パネルの亜種として，配列定数パネル，連想配列定数パネルがある。この二つは入力欄を複数持ち，配列や連想配列を定数として出力できる。配列や連想配列などデータ型については後述する。

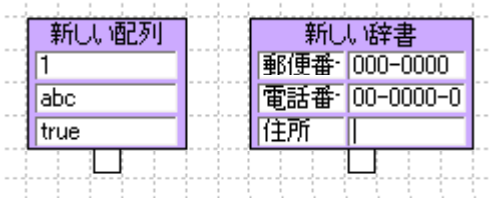

図 4.18 配列定数パネル，辞書定数パネル

4.8.2　演算パネル

受け取ったデータを演算して結果を出力する，最も一般的なパネルである。種類が多いため，四則演算，論理演算，文字列操作，配列操作，条件分岐など，いくつかのグループに分けて部品領域に置かれる。

図 4.19 演算パネル（加算）

4.8.3　高階関数による配列演算パネル

パネル型データを利用するために，いくつかの標準パネルが用意されている。関数型言語の一般的な呼び方では map, foldR, foldL, filter, take, drop, reverse にあたるようなパネルだが，この呼び方のままでは，ゆばの想定する利用者層には親しみにくい。パネルの名前として，map は「全部に」，foldL は「正順まとめ」，foldR は「逆順まとめ」，filter は「抽出」，take と drop は「切出し」，reverse は「逆順配列」という名前にしてある。

関数型言語の foldL/foldR 関数は，初期値を取るかどうかでさらに 2 種類ずつにわかれる。ゆばの「まとめ」パネルは，初期値を入力するための入力端子を持ち，ここにパイプをつなぐかどうかで初期値を使うかどうかを区別する。初期値をとらず，かつ空の配列を渡された場合には，ナル値を出力する。

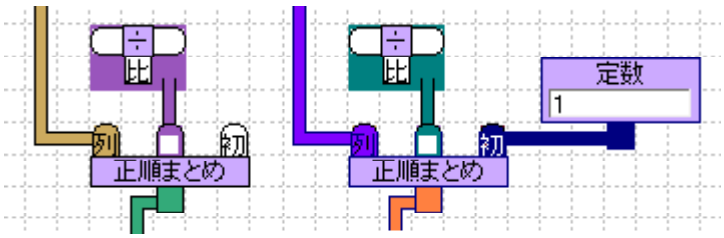

図 4.20 正順まとめパネルに初期値を指定した場合としない場合

take と drop 関数に相当するパネルは「切出し」の 1 種類である。ゆばではパネルが複数の出力をもてることを利用し，先頭側部分配列と残りの部分配列の両方を同時に出力するようにしているからである。

map 関数は 1 引数の関数を引数にとるが，「全部に」パネルは 1 入力のパネルも 2 入力のパネルも引数にとれる。2 入力のパネルを与えて実行した場合，そのパネルには引数として要素の値とキーが与えられる。

これら配列演算パネルに，配列ではなく連想配列を渡した場合，「全部に」と「抽出」は問題なく動作する。「まとめ」と「切出し」が動作するときの要素の順序は不定である。「逆順配列」は何もしない。

4.8.4　適用パネル

パネル型データを実行するためのパネルである。どんな入力数・出力数のパネルデータにも対応できるよう，入力端子数・出力端子数とも可変である。

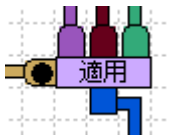

図 4.21 適用パネルで 3 入力 1 出力のパネルデータを適用

入出力数の合わないパネルデータが与えられたときには，

- 実行パネルの入力端子が余るなら，余った右側の端子に入ったデータフローは無視される。
- 実行パネルの入力端子が足りないなら，パネル型データの入力端子が右側から足りない数だけ未配管のままとして実行される。
- 実行パネルの出力端子が余るなら，余った右側の端子からは未配管を示すエラー値が出力される。
- 実行パネルの出力端子が足りないなら，パネル型データの出力端子のうち，右側から足りない数だけの出力が無視される。

4.8.5　記憶スロットパネル

プロジェクトに設定された記憶スロットの読み込みと書き込みのためのパネルである。各記憶スロットについて「読込」と「書込」の 2 種類ずつのパネルが用意されている。読込パネルは 0 入力 1 出力で，書込パネルは 1 入力 0 出力のパネルである。

あるスロットの読込パネルはプログラム上に何枚でも配置できるが，書込パネルは 1 枚しか配置できない。

書込パネルは，結果端子と同じく，プログラム実行に伴ってデータ要求を行う。HTTP リクエストに応じて直接起動されたプログラムの中では書込パネルは動作するが，部品として他のプログラムから呼び出されたプログラムの中では書込パネルは無視される。

4.8.6　モジュールパネル

プロジェクトに属するモジュールはパネルとしてキャンバスに配置できる。

このとき，パネルの形状は 2 種類から選択できる。モジュールの引数端子（差し替え項目）の数だけ入力端子があるパネル（通常型パネル）か，入力端子が 1 つだけのパネル（コレクション入力型パネル）かである。コレクション入力型パネルは，入力を配列または辞書データとして受け取る。

コレクション入力型パネルは，コレクション型データを受け取り，演算の際にはその要素のデータをモジュールの引数端子に渡す。配列データであれば格納順序がそのまま，引数端子の左端からの順番に対応して割り当てられる。辞書データであれば，キーと引数端子の名前を対応させて割り当てられる。

クリエイタの作るパネルは，作成の段階で引数の数が変化する可能性があり，通常型パネルで

あればそのたびにパネルの形状が変わって再配置・再配管を要してしまう。そのような手間を避けることができるのが，モジュールの仕様を変えてもパネル形状が変化しないコレクション入力型のパネルである。

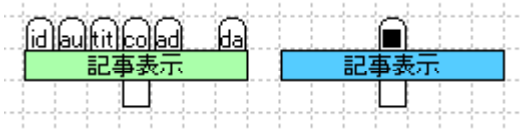

図 4.22 通常型パネルとコレクション入力型パネル

パネルとして，他のプロジェクトのモジュールも使用できる。プロジェクトに他のプロジェクトをインポートすると，部品領域にそのプロジェクト名の分類が新設され，公開すると設定されているモジュールのスタンプがそこに出現する。

他プロジェクトのモジュールを使用した場合，そのモジュールが記憶スロット機能を使っているなら, 間接的に他プロジェクトの記憶スロットの内容を参照することが可能になる場合がある。モジュールを公開する際は，記憶内容を不必要に公開してしまわないよう考慮する必要がある。

4.8.7 リダイレクトパネル

リダイレクト指令型のデータを発生させるためのパネルである。入力として URL またはパネル名を取り，そこへ転送する HTTP レスポンスを生成する。

主な使い方は，POST された値を記憶スロットに格納してから，メイン画面へ自動的に遷移するようなプログラムを書くときである。このような使い方をするには，出口端子に転送パネルを直接つなぐ一方，POST データを処理した結果を書込パネルにつなぐというプログラミングをすることになる。

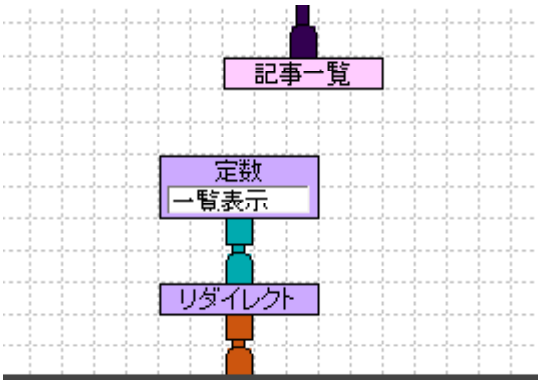

図 4.23 処理結果は記憶スロットに保存され，出力されるのはリダイレクト指令だけ

第5章　実装

ゆばはオンライン開発環境であり，開発者が操作するクライアントと言語処理系が動作するサーバからなるクライアントサーバシステムである。クライアント，サーバ，およびそれらの通信仕様，そしてサーバが使用するデータベースシステムのそれぞれについてどのように構築したかを述べる。

5.1 クライアント

クライアントソフトウェアはすべて Javascript で記述されている。クリエイタが開発環境の URL にアクセスするとクライアントプログラムがブラウザにダウンロードされる。Javascript コードを実行するのは Web ブラウザの標準機能であり，Flash や Java applet と違ってプラグイン等をインストールする必要がない。
クライアントを構成する主要なクラスは図 5.1 のような関係になっている。

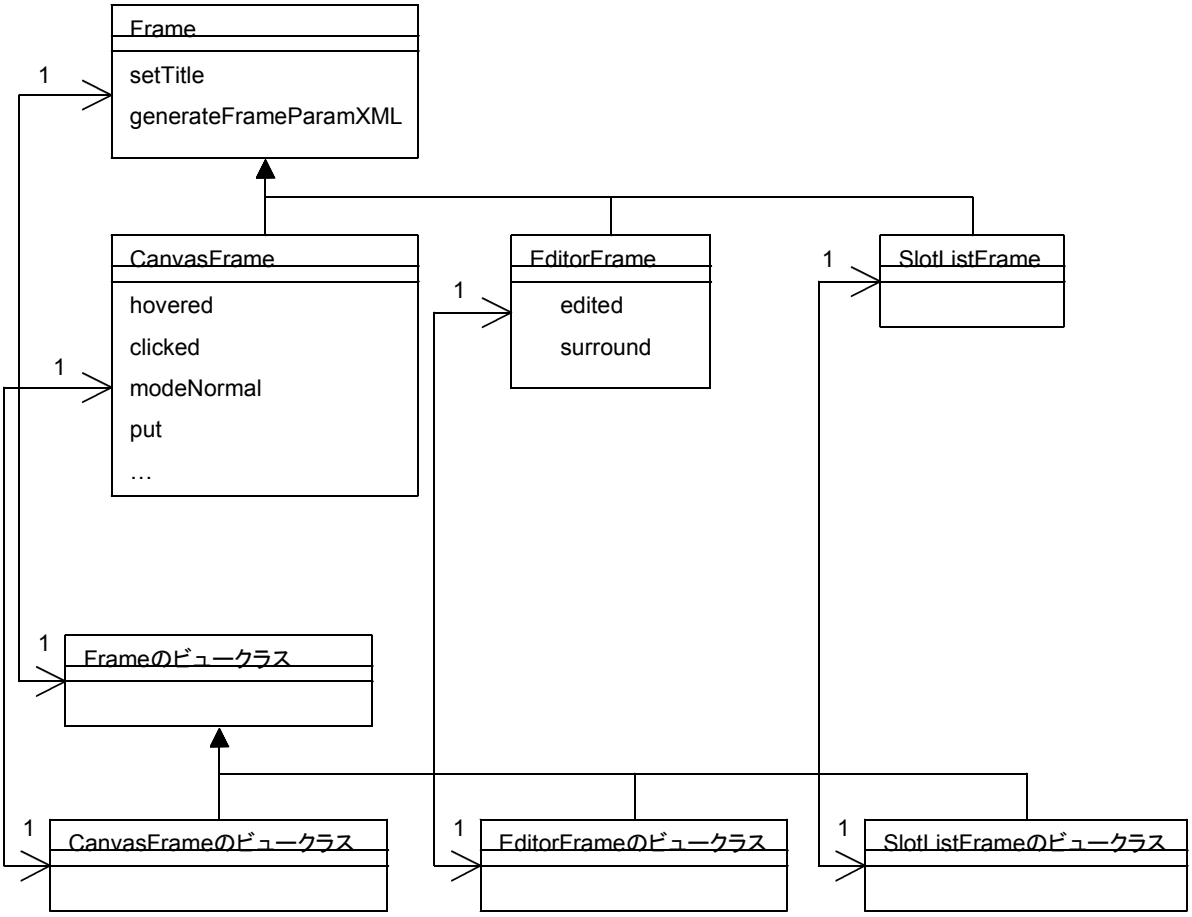

図 5.1 フレーム関連の主要クラス

クライアントのクラス設計は MVC（Model, View, Controller）の分離を徹底している。view に相当するオブジェクトには，HTML の DOM（Document Object Model）オブジェクトをそのまま流用している。そのため view オブジェクトには特にクラス名がついていない。
主要なクラスと，その主要なメソッドについてこの項で解説する。

Frame クラス

キャンバスフレームとエディタをあわせた編集ウィンドウを表す model クラスである。編集ウィンドウに共通するメソッドをまとめてある。

- setTitle メソッド
  タイトルを設定し，自身の view のタイトル表示も変更する。他のモジュールと名前が衝突

しないかもチェックする

- generateFrameParamXML メソッド
  モジュールのタイトルや縦横のサイズ，入出力の数と名前など，モジュールに共通する属性を XML（eXtensible Markup Language）の DOM オブジェクトとして出力する

CanvasFrame クラス

プログラムの編集ウィンドウであるキャンバスフレームを表す model クラスである。キャンバス上のオブジェクトの配置を保持し，それらを操作するメソッドを持つ。

- save メソッド
  プログラムを保存するために図形情報を XML　DOM オブジェクトに変換し，サーバに送信する
- setParams, setReturns メソッド
  引数端子，結果端子の数を変更し，間隔を調整したり自動で名前を付けたりする
- clicked, hovered メソッド
  キャンバスフレームの view オブジェクト上でマウスイベントが発生したとき，その通知を受けるメソッド。イベントが起きたマス目を覆うオブジェクトの clicked, hovered を呼び出す
- clickedOnPanel, clickedOnPipe, clickedOnInput, clickedOnOutput, hoveredOnPanel, … メソッド
  clicked や hoverd によってキャンバスフレーからイベント通知を受けたオブジェクトは，キャンバスフレームのメソッドを呼び返す。オブジェクトがパネルなら clickedOnPanel を，パイプなら clickedOnPipe を，というようにである。これらのメソッドに，実際のイベントへの応答が記述されている。
- modeNormal, modeSelectInput, modeSelectOutput メソッド
  Javascript では，オブジェクトのメソッドを動的に書き換えることができる。キャンバスフレームの操作状態（何も選択されていない／入力端子が選択されている／出力端子が選択されている）によってマウスイベントに対する動作を変更するために，clicked, hovered や clickedOnPanel などのイベント応答用メソッドをすべて差し替えるのが mode*メソッド群である
- put, remove メソッド
  図形がキャンバスに配置されたときや削除されたとき，どのマス目がどの図形に覆われているかを求め，記録するメソッド。また，図形側にも配置されたときの初期化動作があるので，図形オブジェクトの putOnFrame, removeFromFrame を呼び出す
- putPipe メソッド
  入力端子と出力端子が指定されたときに呼び出され，パイプ経路の算出とパイプオブジェクトの生成，さらにその配置まで行う

EditorFrame クラス

テキストの編集ウィンドウであるエディタの model クラスである。

- save メソッド
  テキスト情報を XML DOM オブジェクトに変換し，サーバに送信する
- edited メソッド

エディタの view オブジェクトで内容が変更されたとき呼び出され，model の持つ内部的なテキストデータを更新する

- addParam, removeParam メソッド
  差し替え項目を増減する
- surround メソッド
  カーソル位置または文字列選択範囲の前後に指定された文字列を挿入する。タグ入力ボタンがクリックされたとき呼び出される

SlotListFrame クラス

スロット一覧を表示するウィンドウのクラスである。モジュールを編集する目的のウィンドウではないので，save メソッドは持たない。

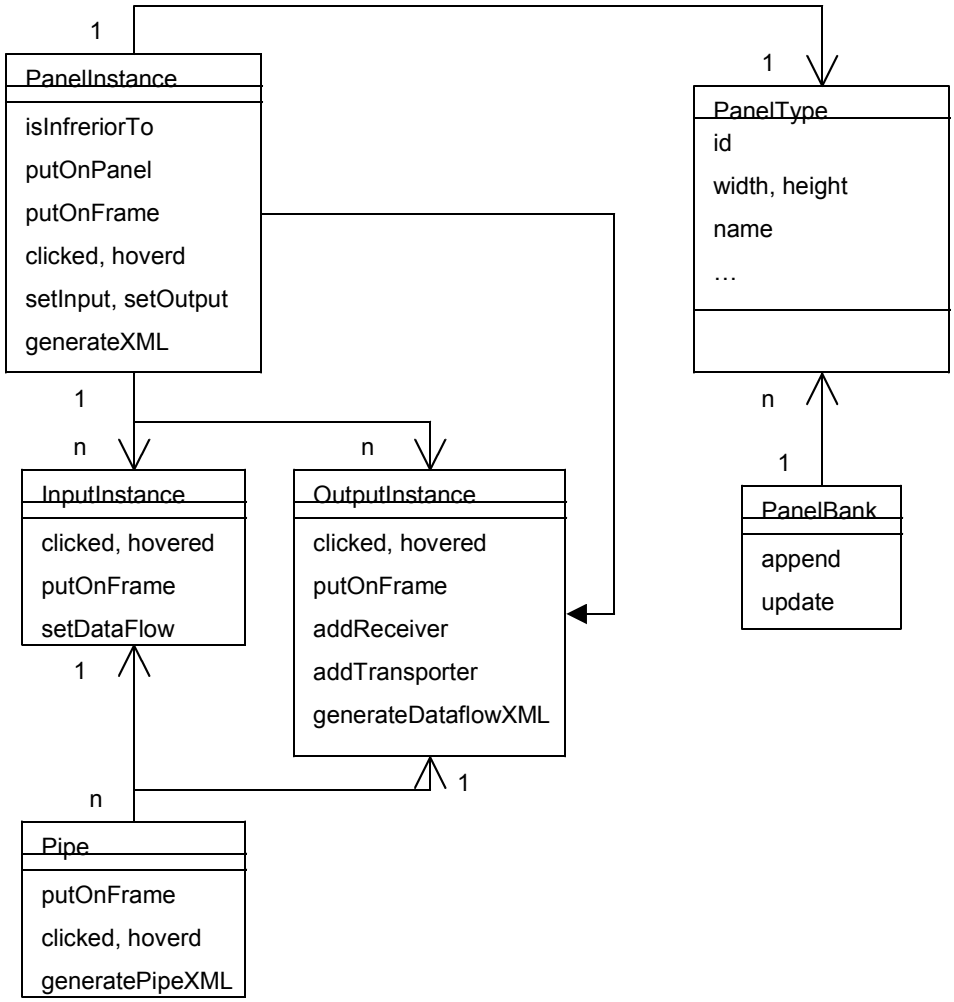

図 5.2 パネル・パイプ関連の主要クラス

PanelType クラス

パネルの種別を表す model クラスである。名称，端子の数と名前，パネルの形状，端子数変更の可否などを保持する。

- setInputCount
  スタンプの端子数増減ボタンがクリックされたときに呼び出される。端子数が変わるのに従い，パネルの形状を再構成する

PanelInstance クラス

キャンバスに配置されたパネルを表す model クラスである。種別（PanelType オブジェクトへの参照），位置，属する端子への参照，配置されたキャンバスへの参照などを保持する。

- isInferiorTo メソッド
  パネルを引数に取り，パイプを出力端子から入力端子方向に辿ってこのパネルから到達できるかを求める。循環参照のチェックの際に呼び出される
- putOnFrame, removeFromFrame メソッド
  キャンバス上に配置された際のフィールド初期化や，削除された際の後片付けをする
- hovered, clicked メソッド
  キャンバス上のマウスイベント通知を受ける
- addReceiver, removeReceiver メソッド
  パネル本体が出力端子として働くときのためのメソッドである。この出力端子から入力を受ける入力端子の一覧に要素を追加／削除する
- addTransporter, removeTransporter メソッド
  パネル本体が出力端子として働くときのためのメソッドである。この出力端子からのデータを運ぶパイプの一覧に要素を追加／削除する
- setOutput, setInput メソッド
  このパネルの端子にパイプが接続されたことの通知を受ける
- generateXML, generateDataFlowXML メソッド
  このパネルの XML　DOM オブジェクトを出力するのが generateXML，このパネルが出力端子として働いているとき，出力端子としての XML　DOM オブジェクトを出力するのが generateDataFlowXML である

InputInstance クラス

入力端子を表す model クラスである。プログラムの結果端子もこのクラスのオブジェクトである。

- putOnFrame, removeFromFrame メソッド
  キャンバス上に配置された際のフィールド初期化や，削除された際の後片付けをする
- hovered, clicked メソッド
  キャンバス上のマウスイベント通知を受ける
- setDataFlow, removeDataFlow メソッド
  パイプがつながったこと／削除されたことの通知を受ける

OutputInstance クラス

出力端子を表す model クラスである。プログラムの引数端子もこのクラスのオブジェクトである。

- putOnFrame, removeFromFrame メソッド

キャンバス上に配置された際のフィールド初期化や，削除された際の後片付けをする
- hovered, clicked メソッド
  キャンバス上のマウスイベント通知を受ける
- addReceiver, removeReceiver メソッド
  この出力端子から入力を受ける入力端子の一覧に要素を追加／削除する
- addTransporter, removeTransporter メソッド
  この出力端子からのデータを運ぶパイプの一覧に要素を追加／削除する
- generateDataFlowXML メソッド
  この出力端子を表す XML DOM オブジェクトを出力する

Pipe クラス

キャンバスに配置されたパイプを表す model クラスである。出力端子への参照，入力端子群への参照，表示色などの情報を保持する。
- putOnFrame, removeFromFrame メソッド
  キャンバス上に配置された際のフィールド初期化や，削除された際の後片付けをする
- hovered, clicked メソッド
  キャンバス上のマウスイベント通知を受ける
- generatePipeXML メソッド
  このパイプを表す XML DOM オブジェクトを出力する

PanelBank クラス

部品領域を表す model クラスである。すべての PanelType への参照を保持する
- append, update メソッド
  パネル種別を新規登録，または変更する。変更は，端子数変更を行った場合におこる

5.2 サーバ

サーバを構築する処理系には，Java ベースのウェブサービス実行環境 Tomcat[1]を採用した。Java/Tomcat の最大の利点は，開発に統合開発環境 Eclipse[2]が利用できることである。

言語処理系のシステムは複雑になり，その構築やデバッグは煩雑な作業である。Eclipse の持つ強力な Java ソース解析・デバッグ機能は複雑なシステムを管理する負担を軽減する。また，Java の徹底したオブジェクト指向は多人数のチームによる開発において，分業をしやすくする。

5.2.1 ModuleStructure

サーバの最も重要な役割は，モジュール情報をクライアントと送受信し，送られてきたモジュール情報を蓄積し，HTTP リクエストに応じて蓄積したモジュールを実行することである。

この役割の中で，サーバが扱うモジュールの情報は次の 3 つの形態を取りうる

[1] http://tomcat.apache.org/
[2] http://www.eclipse.org/

- クライアントと送受信するとき：**XML ドキュメント**
- DB に保存したとき：**DB テーブル上のレコード**
- モジュールを実行するとき：**演算メソッドを持った Module クラスのオブジェクト**

サーバは，モジュールを扱う際にこの 3 つの形態の間で相互に変換を行っている。

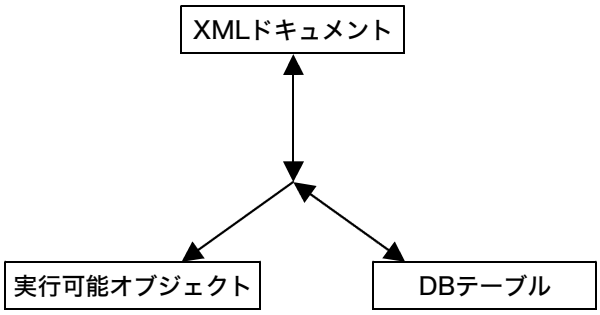

図 5.3

この 3 つの形態を変換する中間的な形式として，ModuleStructure というクラスを経由している。ModuleStructure はモジュールを表現するための文字列情報だけを保持するクラスで，Module クラスよりも高速にインスタンスを生成できる。。

この ModuleStructure を中心にして，

- ModuleStructure オブジェクトを XML ドキュメントに変換する
- XML ドキュメントを ModuleStructure オブジェクトに変換する
- ModuleStructure オブジェクトを分解して DB テーブルに保存する
- DB から読み込んだデータで ModuleStructure を構築する
- ModuleStructure オブジェクトから Module オブジェクトを構築する

という 5 種類の操作をするのがサーバの基本的な動作となっている。

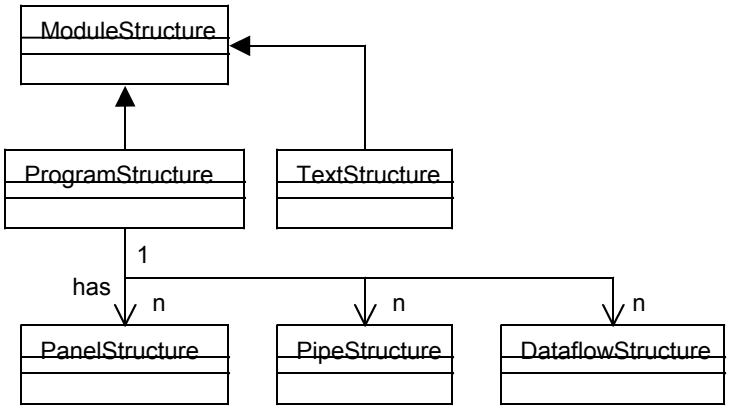

図 5.4

プログラムの情報を保持する ProgramStructure クラスは，複数のパネルや複数のパイプなどを格納するため，PanelStructure 等の別の中間形式クラスのコレクションを持っている。

5.2.2 Module

モジュールを表現するための，演算メソッドを持ったクラスである。このクラスのインスタンスは，getResult(int　n)メソッドをコールされると，格納されているモジュールを実行し，その結果を返す。

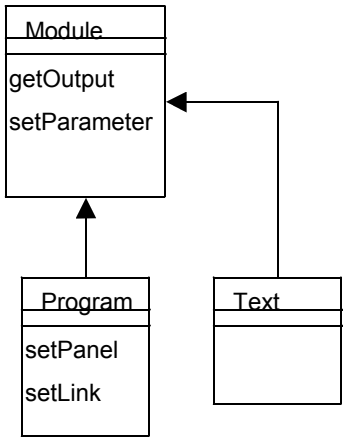

図 5.5

Program クラスでは，結果端子や引数端子を通常のパネルの入力端子と出力端子と同じように扱うため，SystemPanel という仮想的なパネルを表すクラスのインスタンスを 1 つを持っている。SystemPanel はプログラムの結果端子を入力端子，引数端子を出力端子とするパネルである。プログラム開始時，出力端子にはプログラムに与えられた引数データが格納されている。そして，Program の getResult メソッドが呼ばれると，SystemPanel の入力端子がデータ要求を出しはじめる。これによって，言語定義の節で解説したプログラム実行が行われるようになっている。

5.2.3 Panel

Panel クラスはパネルを表現する。実際にはキャンバスに配置されるパネルだけでなく，前項で述べた SystemPanel のような仮想的なパネルも表現対象に含まれる。仮想的なパネルは SystemPanel のほかにもう 1 種，ErrorPanel というクラスがある。これは，Program の構築に必要な Panel のサブクラスをクラスローダが読み込めなかったときに生成される仮想的パネルである。ErrorPanel は常にエラー値を出力する。

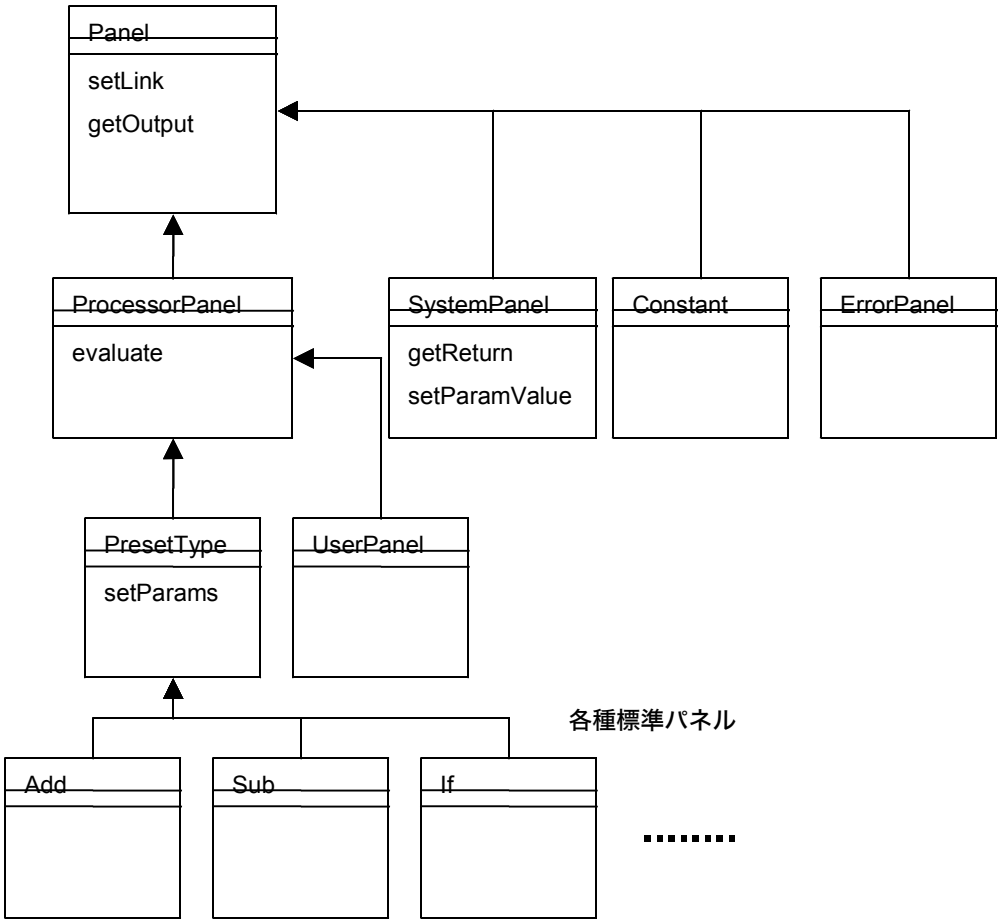

図 5.6 サーバにおけるパネル関連のクラス構造

実際にキャンバス上に配置できるパネルのうち，入力を取って演算し，出力をするものの親クラスが ProcessorPanel である。この下には，標準で用意されている演算パネルすべての親 PresetType と，モジュールを実行するパネル UserPanel がある。

PresetType のサブクラスは多数あるため，Program のイニシャライザにはこれらすべてを呼び分けるコードは記述していない。パネル種類ごとのクラス名が DB に保存されており，ここから読み出したクラス名でクラスローダに呼び出させている。

Constant パネルは，プログラム実行前から出力端子の値が決まっているパネルである。その値は DB に保管されており，保管の形式については次項で詳述する。

5.2.4 Var

Var は，動的データ型体系であるバリアント型を格納するクラスである。

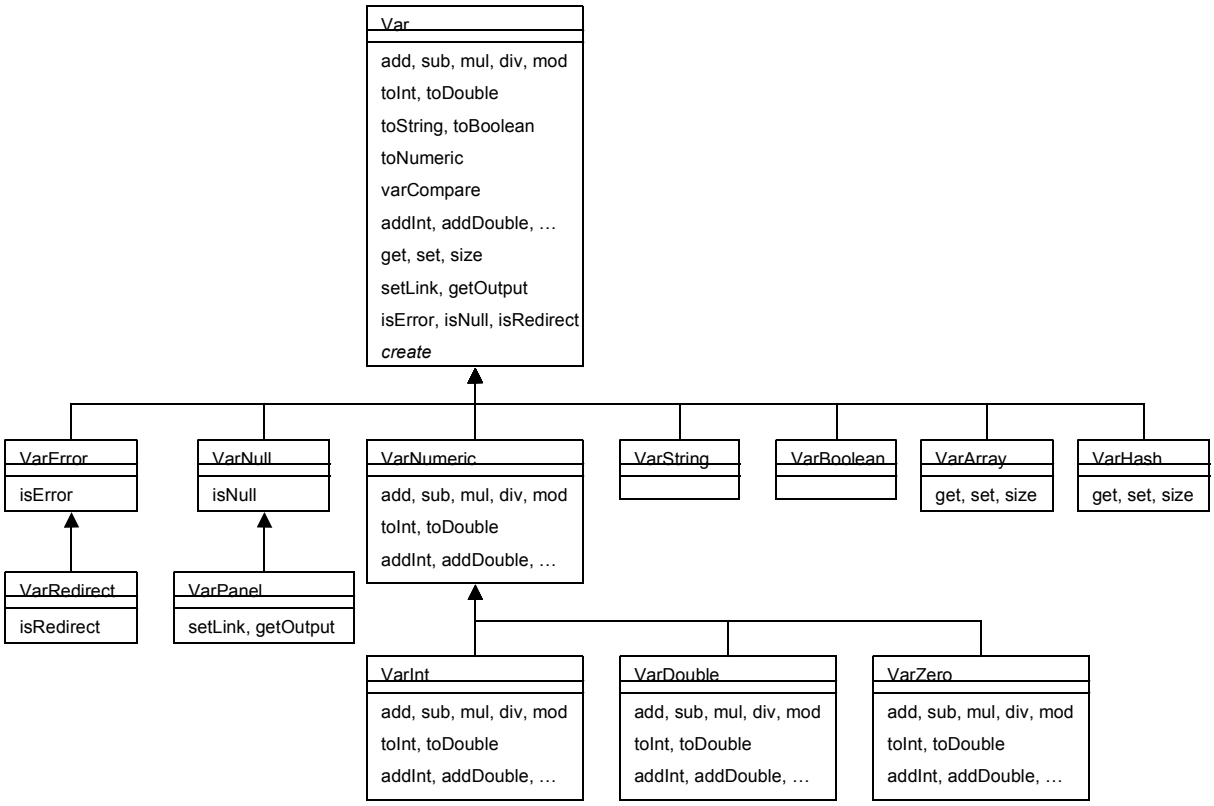

図 5.7 バリアント型のクラス構造

Var 型各クラスには破壊的なメソッドがない。例えば加算を意味する add(Var　v)メソッドは，そのインスタンスの値を引数の値だけ増やすのではなく，そのインスタンスの値と引数の値を加算した値の，新しい Var インスタンスを返す。

Var インスタンス同士の四則演算では，クラスの組み合わせによって結果の値のクラスが違う。整数型（VarInt）データ同士の加算や積算なら結果も整数型だが，整数型データと実数型データ（VarDouble）との演算では結果は実数型になる。数値型以外の型のデータとの演算では，データを数値型（整数型か実数型）に変換したものが演算対象となり，上記のルールで結果の型が決まる。

このような演算ルールをメソッドの多態性によって実装するため，Var は特有の演算メソッド群を持っている。

VarNumeric toNumeric()メソッド

数値型に変換したデータを作成して返すメソッドである。VarInt や VarDouble のこのメソッドは，this を返すだけとなっている。

Var add(Var v) メソッド

演算の例として加算を挙げる。バリアント型データを取って，自分との加算結果のバリアントデータを返すというものである。このメソッドの内容として，Var 型には

```
return this.toNumeric().add( v );
```

とだけ記述してある。VarNumeric 各クラスが実質的な加算メソッドを持っているため，数値型でないクラスのデータはここで自動的に数値型へ変換した上で演算を実行するようになっている。VarNumeric クラスは add は abstract メソッドとしてオーバーライドしており，サブクラスに実質的な演算メソッドの実装を強制している。

数値型クラス，例えば VarInt において add には

```
return v.addInt( this.value );
```

とだけ記述されている。ただし，this.value は VarInt の格納データを表す int 型フィールドである。加算する相手に，自分の数値を渡して「整数との加算をしろ」というメッセージを送る仕様になっている。最終的な加算は addInt の中に記述されており，VarInt においてなら

```
return new VarInt( value + this.value );
```

と，整数型を生成して結果を返すことが記述されている。加算数と被加算数が互いに相手を呼び合うという枠組みで，演算の型ルールを実装しているのである。乗算，減算，剰余も同様の仕組みで実現されている。除算だけは，整数同士でも結果は実数型でないといけないのでこの方式はとらず，単純に double 型に変換して除算を実行しているだけである。

int varCompare(Var v)メソッド

バリアントデータ同士の数値比較を行うためのメソッドが varCompare である。数値として解釈したとき，引数が自分より大きいなら負の数を，自分と等しいなら 0 を，自分より小さいなら正の数を返すという仕様である。

この比較演算も，add と同じく左辺値と右辺値が相互に呼び合う方式で実装されている。

5.3 データベース

サーバが永続的に保持しているすべきデータとしては，

- クリエイタの作成したモジュール
- 記憶スロットの内容

が主要なものである。これらのデータの保持には，RDBMS（Relational Data Base Management System）を利用することとした。

ゆばは不特定多数のクリエイタやユーザが利用することを前提としたシステムである。クリエイタの作成したモジュールや，ユーザがシステムを利用するときに生成される記憶スロットデータをファイルとして保存した場合，ファイルシステム上は管理者にとって内容不明なファイルであふれてしまい整理がつかなくなることが考えられる。そこで，データを絞り込んだり整列したりしての操作が容易な，RDBMS により管理された DB を利用すべきと考えた。

RDBMS としては, 実績が多く, オープンソースで無料での利用が可能かつ実績の優れた MySQL [1] Ver.4.1 を採用した。

5.3.1 バリアント型データの保存

DB に記憶スロットや定数パネルの内容であるバリアント型データを保存するには，格納され

[1] http://www.mysql.com/

たデータの種類や長さが一定でないというデータ構造を合理的に保存するテーブル構造が必要である。

これを，次のような構造のテーブル var で実現した。

カラム名	id	type	intData	doubleData	stringData	arrayId	hashKey
データ型	int	int	int	double	text	int	text

表 5.1 var のテーブル構造

type でデータ型を区別する。プリミティブ型のデータは intData, doubleData, stringData のいずれかのカラムを使って格納する。真偽値型は intData カラムを流用し，ナル型はどのカラムも使用しない。

配列型を格納するときは，type に配列を示す値（7）をセットしただけのレコードをまず格納し，これを親レコードと呼ぶことにする。次に各要素値を先頭から順番に，別のレコードとして格納していく。このとき，要素値のレコードは arrayId カラムに親レコードの id をセットするようにする。

連想配列型を格納するときは，type に 8 をセットした親レコードを格納してから，各要素値の arrayId に親レコードの ID を，hashKey にキー文字列を設定して格納する。

5.4 データ交換

サーバとクライアント間の通信は，HTTP で行われる。クライアントを記述した言語の Javascript は，HTTP 通信以外の通信機能を持っていないためである。HTTP 上のデータ交換について述べる。

5.4.1 文字コード

通信路上に限らず，ゆばのシステムではクライアント・サーバ・DB とも，文字列のエンコード形式として UTF-8[19]を用いている。Javascript の HTTP 通信オブジェクトである XMLHttpRequet がエンコードとして UTF-8 を前提にしていること，Unicode 文字セットを表現できるため将来ゆばを国際化するときにも使えることが理由である。

5.4.2 交換形式

サーバ／クライアント間のデータ交換は，XML で行っている。プログラムの図形情報のように複雑な構造を持ったデータを通信路に乗せるには，サーバとクライアントの双方で利用可能なシリアライズ方式を利用するのが合理的であり，Java でも Javascript でも利用可能なのが，XML だからである。

DTD（Document Type Difinition）ファイルとの照合で不正な形式のデータをはじくことができるのも XML の利点である。

第6章　サイト作例

ゆばを利用することにより，ウェブシステム構築が容易になることを示す。その例として，ウェブ掲示板を作成する。比較対象として，現在ウェブプログラミングで幅広く用いられている PHP 言語+MySQL[1]で構築した場合のコードを示す。

6.1　サイト仕様

作例サイトはウェブ掲示板であり，不特定多数の利用者が投稿した記事を時系列に並べて表示することができるウェブシステムである。

6.1.1　記事の仕様

記事は，ID 番号，投稿者名，投稿者メールアドレス，題名，本文，投稿日時というデータを持つ。各データ内に HTML タグが含まれていても，表示される際にはエスケープされてタグとしての意味は持たない。

6.1.2　画面の構成

記事を，新しいものから順に 10 件表示する。より古い 10 件・より新しい 10 件を表示するページへのリンクもある。

画面上部には記事の入力欄があり，ここで新しい記事を投稿できる。

記事を投稿する際，本文だけは 1 文字以上入力されていないといけない。投稿日時は自動的に記録される。

6.1.3　セキュリティ

XSS（CROSS Site Scripting）[2]，SQL インジェクション[3]，CSRF（Cross Site Request Forgery）[4]の各攻撃手法に対して脆弱でないものとする。

---

[1]この組み合わせを，サーバ OS に Linux，ウェブサーバに Apache を採用したシステムで使用する開発環境を，Linux, Apache, MySQL, PHP の頭文字を取って LAMP と呼ぶ。

[2]ユーザが書き込む文章中にタグを書き込むことができ，それがウェブページにそのまま表示されてしまう場合に，悪意を持った Javascript コードを書き込むことで閲覧者のブラウザ上でそのコードを実行させるという攻撃手法。

[3]引用符，セミコロンなど SQL クエリ文の中で意味のある文字を適切にエスケープ処理せずに DB に登録使用とするシステムに対し，このような文字を含む記事を投稿して SQL 分を誤作動させる攻撃手法。

[4]HTML のフォーム情報は，それがどのサイトに設置されていたフォームであっても送信されれば送信先のウェブサーバは受け付ける。表示したとたんにフォームを投稿するように設定されたウェブページを作ることで，閲覧者に意図しないフォーム投稿をさせるという攻撃手法。

6.2 PHP+MySQL による作例

前項の仕様を満たすウェブシステムを PHP+MySQL で構築した。

6.2.1 DB 定義

```
CREATE TABLE `sampleBBS`.`articles` (
    `id` INT NOT NULL AUTO_INCREMENT ,
    `subject` TEXT NOT NULL ,
    `author` TEXT NOT NULL ,
    `address` TEXT NOT NULL ,
    `content` TEXT NOT NULL ,
    `timestamp` VARCHAR( 18 ) NOT NULL ,
    PRIMARY KEY ( `id` )
)
```

6.2.2 スクリプト

```
<?php
// データベース接続を試行
$connection =  @mysql_connect( "localhost", "***",  "***" );
if(!$connection or !@mysql_select_db( "sampleBBS", $connection ) ){
    // データベース接続失敗
    print( "データベースに接続できません" );
    exit;
}
mysql_query("set names utf8");

// CSRF 対策のトークンを生成
$token = md5(
    $_SERVER['REMOTE_ADDR'].
    $_SERVER['HTTP_USER_AGENT']."_token" );

// GPC マジッククオートが有効だった場合，無効化
if (get_magic_quotes_gpc()) {
    function stripslashes_deep($value){
        return is_array($value)
        ? array_map('stripslashes_deep', $value)
        : stripslashes($value);
    }
    $_POST = stripslashes_deep( $_POST);
    $_GET = stripslashes_deep( $_GET);
```

```
}

// 投稿があった場合受け付ける
if( $_POST["content"]!=""  && $_POST["token"]==$token ){
    $author= mysql_real_escape_string( $_POST["author"] );
    $address= mysql_real_escape_string( $_POST["address"] );
    $subject= mysql_real_escape_string( $_POST["subject"] );
    $content= mysql_real_escape_string( $_POST["content"] );
    $timestamp = date("y-m-d H:i:s");

    $SQL =<<<DOC_END
    insert into articles
        (author, address, subject, content, timestamp)
    values
        (¥"$author¥", ¥"$address¥", ¥"$subject¥",
         ¥"$content¥", ¥"$timestamp¥")
DOC_END;
    mysql_query($SQL);

    // 画面をリロード
    header( "location:http://".
        $_SERVER["SERVER_NAME"]."/sampleBBS.php" );
    exit;
}

// 何番の記事から表示するか決定
$start = $_GET["start"]+0;
$start = max( 0, $start );
$current=$start + 1;
$prev=$start - 10;
$next=$start + 10;

print <<<DOC_END
<HTML>
<HEAD>
    <TITLE>一覧表示 - サンプル掲示板</TITLE>
    <style> p{margin:0px;} </style>
    <meta http-equiv="charset" value="utf-8">
</HEAD>
<BODY>
<H3>サンプル掲示板</H3>
<HR/>
<FORM action="sampleBBS.php" method="POST">
<p>お名前 <input type="text" name="author"/>
```

```
メールアドレス <input type="text" name="address"/></p>
<p>記事タイトル <input type="text" name="subject"></p>
<p>本文</P>
<p><textarea cols=80 rows=7 name="content"></textarea></p>
<input type="hidden" name="token" value="$token"/>
<p><input type="submit" value="投稿"></p>
</FORM>
<hr/>
<p> $current 番目に新しい記事から表示しています <a href="?start=$prev">
新しい 10 件へ</a> <a href="?start=$next">古い 10 件へ</a></p>
<hr/>
DOC_END;

// 記事一覧取得
$SQL = "select * from articles order by id desc limit $start,10";
$articles = mysql_query( $SQL );

while( $rows = mysql_fetch_array( $articles ) ){

    // 各記事表示
    $id = $rows["id"];
    $author = htmlspecialchars( $rows["author"] );
    $address = htmlspecialchars( $rows["address"] );
    $subject = htmlspecialchars( $rows["subject"] );
    $content = preg_replace(
        '/¥r¥n|¥r|¥n/s', "<BR>",
        htmlspecialchars( $rows["content"] ) );
    $timestamp = $rows["timestamp"];

    print "<p>$id : <b>$subject</b></p>";
    if( $address )
        print "<p>投稿者：<a href=¥"mailto:$address¥"> $author </a> 投
稿日時：$timestamp</p>";
    else
        print  "<p><font  color=¥"gray¥">投稿者：</font>$author  <font
color=¥"gray¥">投稿日時：</font>$timestamp</p>";
    print "<p>$content</p><hr/>";
}
print<<<DOC_END
<p>sampleBBS</p>
</body>
</html>
DOC_END;
?>
```

6.2.3 実行結果

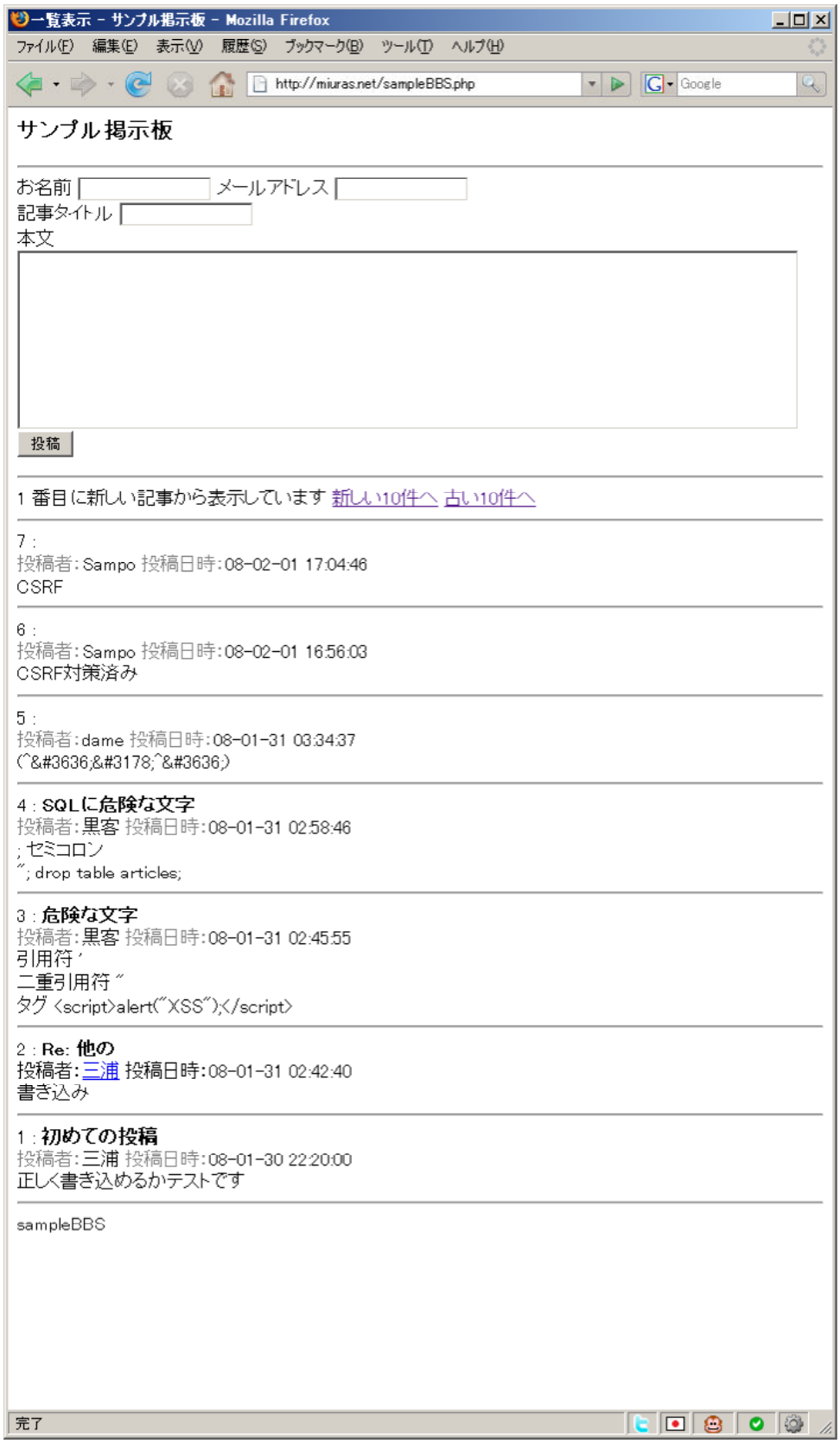

図 6.1 PHP 版掲示板システムの実行結果

6.2.4　作例の評価

本作例は 1 本のテーブルと，1 本のスクリプトにより構成されている。テーブル定義とスクリプト記述として合計約 3.2 キロバイト（注釈行を除く）のテキストデータを入力する必要があった。

このほか，PHP と MySQL をインストールして稼働するよう設定する手間がかかっている。

セキュリティ要求を実現するために get_magic_quotes_gpc，mysql_real_escape_characters，md5，htmlspecialchars などの関数を適切な位置で使用する必要があり，これらの使用の背景となる知識を習得する必要もあった。

6.3　ゆばによる作例

PHP+MySQL による作例と同じく 6.1 の仕様を満たすウェブシステムをゆばで作成した。

6.3.1　記憶スロット

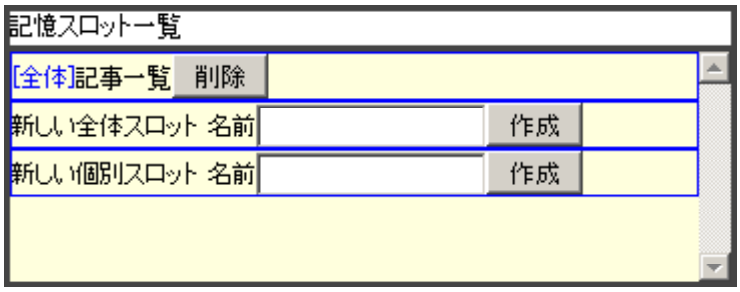

図 6.2 掲示板システムでは記憶スロットを 1 個使う

6.3.2　テキスト

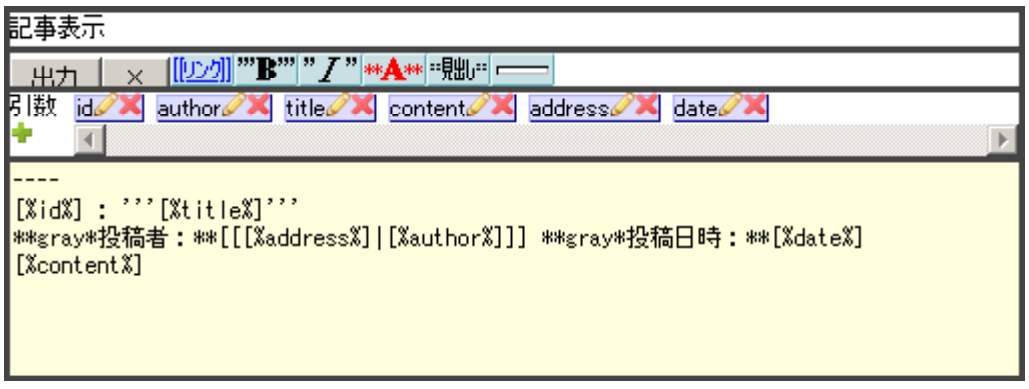

図 6.3 記事一本ごとの表示スタイルを定義

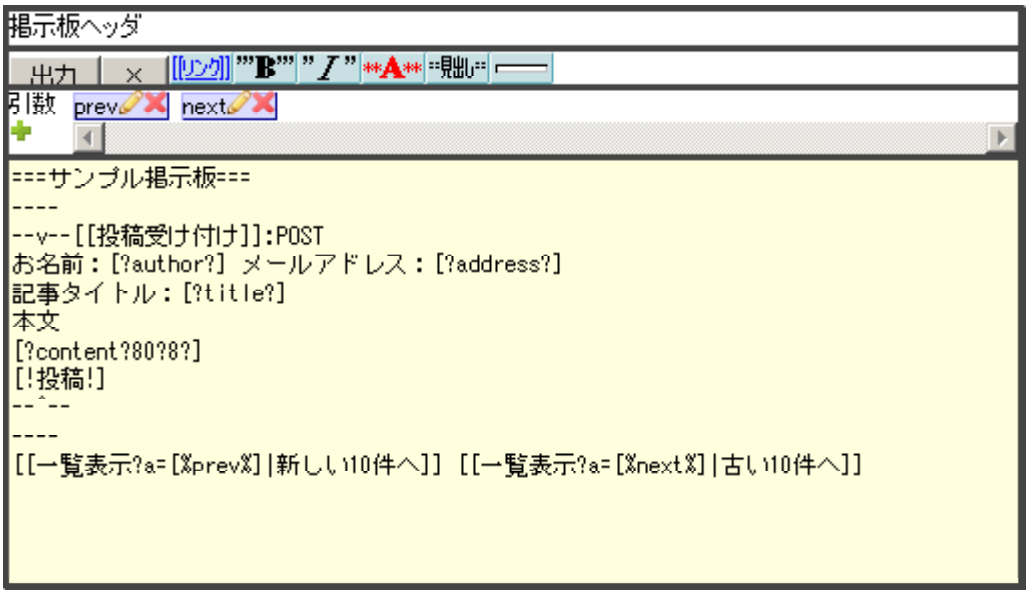

図 6.4 記事一覧ページの先頭表示を定義。投稿フォームも含む

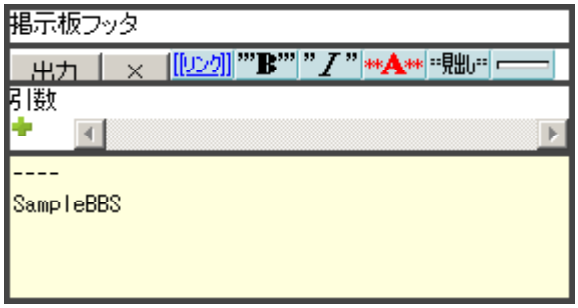

図 6.5 記事一覧ページの末尾表示を定義

6.3.3 プログラム

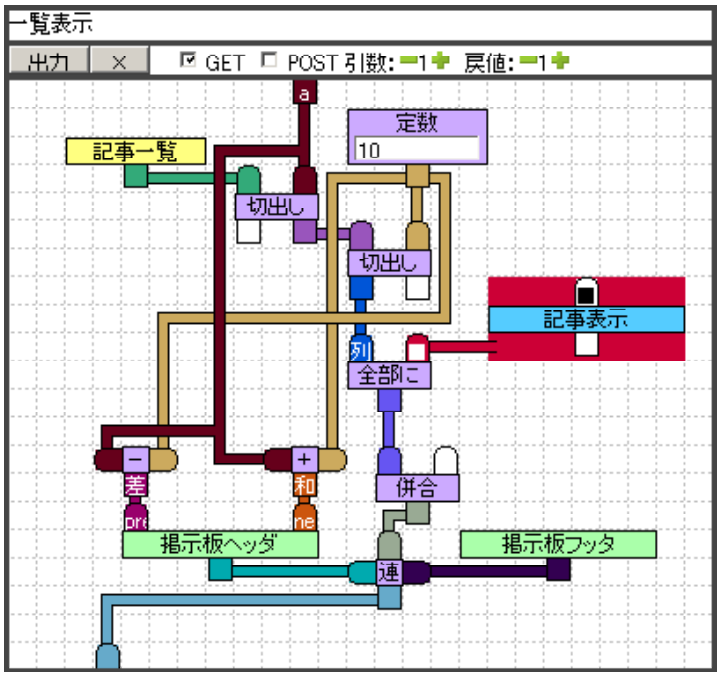

図 6.6 記事を 10 件ごとに切り出して表示するプログラム

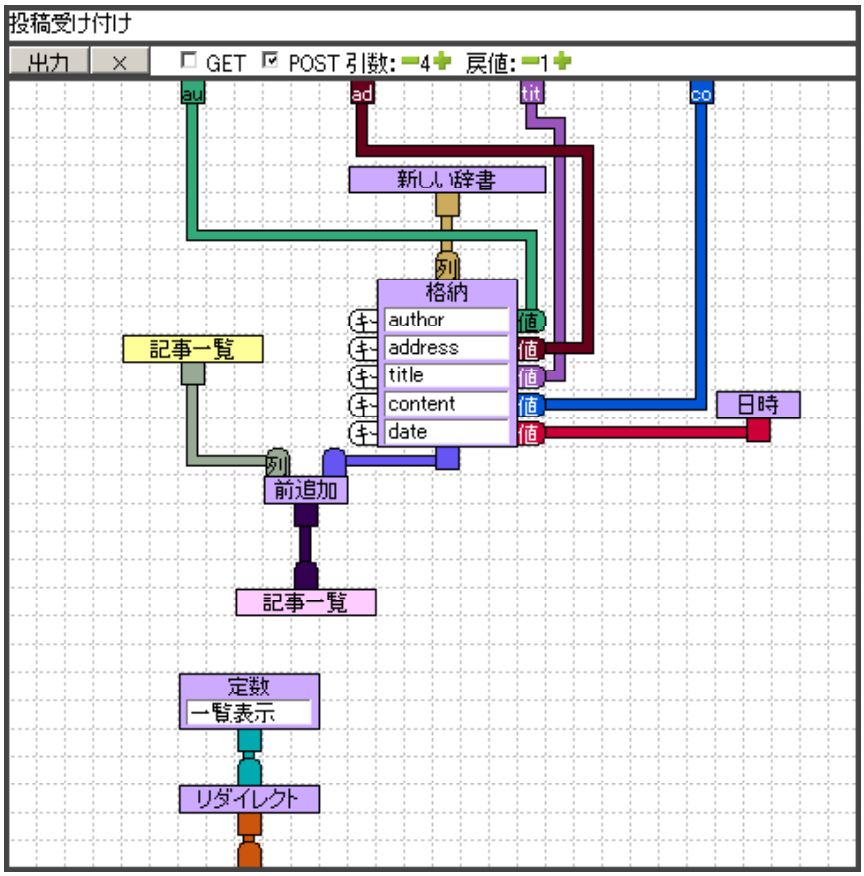

図 6.7 投稿フォームから投稿された記事を記憶スロットに保存するプログラム

6.3.4 実行結果

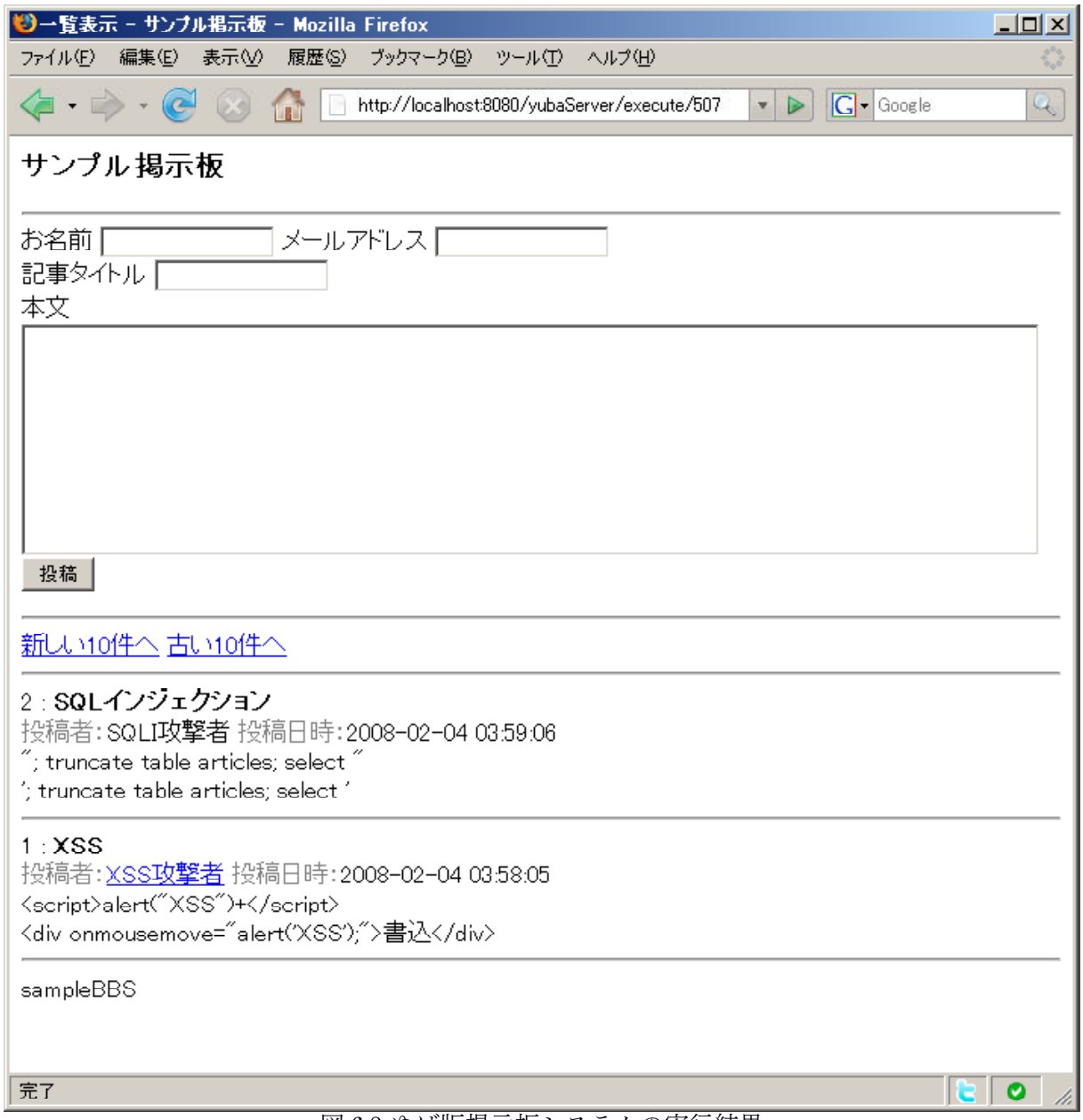

図 6.8 ゆば版掲示板システムの実行結果

6.3.5 作例の評価

プログラムを 2 つ作成し，その作図のためにパネルを 20 枚，パイプを 25 本配置した。パネルは 1 枚配置するのにスタンプ選択と配置位置選択の 2 クリック，パイプは 1 本配置するのに入力端子と出力端子を選択するのに 2 クリックを要する。格納パネルの大きさを変えるのに 4 クリックと，引数端子を増やすために 5 クリックかけたことを合わせても，99 クリックでプログラムは作成できた。セキュリティ面の要求を満足させることを目的としたコーディングはまったくしていないが，要求は問題なく満たされている。

一方，打ち込んだテキストの量は 442 バイトで，しかも実際のタイピング量は入力支援ボタンによりこれよりも少なく済んでいる。

99 クリックと，442 バイト以下のタイピングによって，PHP ＋ MySQL では 3.2 キロバイトのタイピングを要したシステムと同等のシステムが作成できたことになる。しかも，サーバの設定やプログラムのアップロードなどの手間はまったくかかっていない。

プログラムコードの可読性に関しては，非常に主観的な評価となるため，どちらが勝ると一概には言えない。しかし，コード量の少なさが目を通すべき範囲も小さくしている点は指摘できる。

また，ビジュアルプログラミングコードにおいては，テキストプログラミングコードではできなかった，データの流れを指で差しながら追うという直感的な読解が可能であるという点も指摘できる。

第7章　結語

著者らの開発したオンラインビジュアルプログラミング言語処理系「ゆば」の言語定義，動作仕様，および実装について解説し，ゆばを用いることによってウェブプログラミングがどの程度易しくなるのかを，既存のウェブプログラミング環境である PHP+MySQL と比較することで示した。

比較の結果，同等の仕様をみたすウェブシステムを既存環境に比べ大幅に少ない労力でプログラムでき，かつ，環境設定などプログラム公開のための作業も必要ないという優位性を示すことができた。

可読性に関しても，読むべきコードの少なさ，直感的な読解方法がとれる点などにおいて優位性を示すことができた。

以上から，ゆばはウェブプログラミングを既存の処理系を用いるよりも簡単にすると考えられた。このことは，ウェブシステムのアイディアを実現するための敷居を下げることに大きく貢献すると期待される。

参考文献

[1] Tim O'Reilly, "What Is Web 2.0". O'Reilly, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html （2008 年 1 月 27 日）, 2005.

[2] Ward Cunningham, "Wiki History". Cunningham & Cunningham, Inc., http://c2.com/cgi/wiki?WikiHistory （2008 年 1 月 27 日）.

[3] W. Frawley and G. Piatetsky-Shapiro and C. Matheus, "Knowledge Discovery in Databases: An Overview". *AI Magazine*, Fall 1992, pp. 213-228.

[4] Parallax, Inc., "Scribbler Help". The Official Home of the Scribbler Robot, http://www.scribblerrobot.com/Help-v1.0a/english_help.html （2008 年 1 月 27 日）, 2006.

[5] Wayne Citrin, Michael Doherty, Benjamin Zorn, "The Design of a Completely Visual Object-Oriented Programming Language". *Visual Object-Oriented Programming* (M. Burnett, A. Goldberg & T. Lewis, eds.) Prentice-Hall, New York, 1994.

[6] Kimura, T.D., Choi, J.W., and Mark, J.M., "Show and Tell A Visual Programming Language", *Visual Computing Environments*, E.P. Glinert (ed.), IEEE Computer Society Press Tutorial, Washington, D.C., 1990, pp397404.

[7] Andescotia LLC., "Marten & trade IDE 1.3", Andescotia Software, http://www.andescotia.com/products/marten/ （2008 年 1 月 27 日）, 2005.

[8] Daniel D. Hils, "DataVis: A Visual Programming Language For Scientific Visualization", Proceedings of *the 19th annual conference on Computer Science*, ACM, 1991.

[9] Kimura, T.D., "Hyperflow: A Uniform Visual Language for Different Levels of Programming", Proceedings of *the 1993 ACM conference on Computer science*, ACM, 1993.

[10] National Instrumnts, "LabVIEW Web Services", http://software.oit.pdx.edu/math/labview/www/services.htm （2008 年 1 月 27 日）, 2002.

[11] Miller Puckette, "Pure data", Proceedings of *the International Computer Music Conference,* International Computer Music Association, 1997, pp. 224-227.

[12] Cycling '74., "Max/MSP", http://www.cycling74.com/products/maxmsp （2008 年 1 月 27 日）, 2008.

[13] Apple Inc., "Graphics & Imaging Quartz Guides", http://developer.apple.com/documentation/GraphicsImaging/Quartz-date.html （2008 年 1 月 27 日）, 2007.

[14] Squeak foundation, "Squeak", http://www.squeak.org/（2008 年 1 月 27 日）.

[15] NTT コミュニケーション科学基礎研究所, 「Visucuit びすけっと」, http://www.viscuit.com/ （2008 年 1 月 27 日）.

[16] Ken Kahn, "ToonTalk - Making programming child's play", http://www.toontalk.com/ （2008 年 1 月 27 日）, 2007.

[17] Stagecast Software, Inc., "Stagecast", http://www.stagecast.com/（2008 年 1 月 27 日）, 2008.

[18] 橘　純也, 「ユーザに型を意識させない型システムの設計」, 2007 年度早稲田大学理工学部コンピュータ・ネットワーク工学科卒業論文, 2008.

[19] F. Yergeau, "UTF-8, a transformation format of ISO 10646", IETF RFC 3629, 2003.

謝辞

本研究を進めるにあたり多大なアドバイスをいただいた筧捷彦教授に深く感謝します。
そして，共同研究者である佐藤幸弘君，伊藤瑞貴君，稲垣拓海君，柏瀬博之君，橘純也君，田原旬君の協力により 1 年間という短期間での研究完成をみることができたことを付言します。

付録

ゆばのダウンロード

ゆばは，自由にダウンロードしてホストマシンにインストールし運用することができる。ダウンロードには SVN を用いる。

リポジトリアドレス

```
http://giren.kake.info.waseda.ac.jp/svn/yuba
```

ディレクトリ

```
yubaServer
```

ゆばのインストール

OS を問わず，JSE, Tomcat, および MySQL の動作する環境でインストールができる。

JSE

バージョン 5.0 以降をインストールする。

Tomcat

バージョン 5.5 系列をインストールする。

MySQL

バージョン 4.1 以降をインストールする。アクセスアカウントとして，"root"（パスワード:"tonyuniru"）を開設する。安全のため，このアカウントは localhost からの接続のみに解放することを推奨する。

localhost に root/tonyuniru のアカウントで接続する設定以外で運用する場合は，controller.database.YubaDB.java を書き替えることで対応できる。

ゆば

Tomcat の webapps ディレクトリ内に新しいディレクトリを作成してそこに展開する。展開したすべての Java ソースファイルをコンパイルする。

controller.database.Initializer.class を実行するか，memo.itoh.txt の内容を SQL クエリとして MySQL に実行させることで，データベースに必要なテーブル定義が行われる。

ゆばの起動

インストールが正常に行われた状態で Tomcat を起動すると，ゆばが使用可能になる。ウェブアクセスするための URL は，

```
http://ホスト名:8080/ゆばのディレクトリ名/
```

である。