soyText - プログラムとデータを一緒に同期して どこでも動く Web アプリケーションフレームワーク

長 慎也†

本発表では，Web アプリケーションフレームワーク soyText を提案する．Web アプリケーションは，ネットワークに接続していないときや，Web サーバに障害があったときに利用できなくなる，という問題点がある．そこで，soyText では，複数のサーバ，あるいはローカルな環境でまったく同じ環境を稼働させられる仕組みを実現した．soyText は，フレームワーク上で動くプログラムおよびそれらのプログラムが利用するデータを，単一の組み込みデータベース上に一緒に保存している．それらのプログラムおよびデータを他のコンピュータ上にある soyText のシステムと同期することによって，複数のコンピュータ上でまったく同じ環境を再現することができる．

soyText - The Web application framework, runs on everywhere by synchronizing both programs and data

Cho Shinya †

This presentation proposes ”soyText”, Web application frame work. Web applications are not available when network is disconnected or servers failed. soyText solves this problem by copying server environment into client computers. Each system of soyText has single embedded database that contains all program and data needed to run the application. Two or more systems of soyText can easily ”sync” their database to work with the same environment between servers and clients.

1. はじめに

筆者は，これまで Web ブラウザ上で動くプログラミング環境（Web-bases IDE，以下 WBI と呼ぶ) を作って授業で活用してきた．

WBI の利点として，次のようなことがあげられる．

- 学習者による環境のセットアップが不要で，大人数の授業におけるサポートの軽減ができる
- 学習者間のプログラムの共有が容易である，他の人の作品を見たり評価したりすることによってモチベーションをあげることができる
- 学習環境の不具合やバージョンアップを Web ブラウザを通じて動的に行うことができる

これまでに，PHP の開発環境[1)](**図 1**) や JavaScript と Web API を用いた開発環境[2)](**図 2**) などを WBI で作り，実際の授業に用いてきた．

本発表では，これらの実習環境に適した Web フレームワークを提案する．

図 1 PHP の WBI
Fig. 1 WBI in PHP

2. WBI への要件

ここでは，これまでの WBI の開発を通じて得られた，システムがもつべき要件をまとめる．

† 明星大学
Meisei University

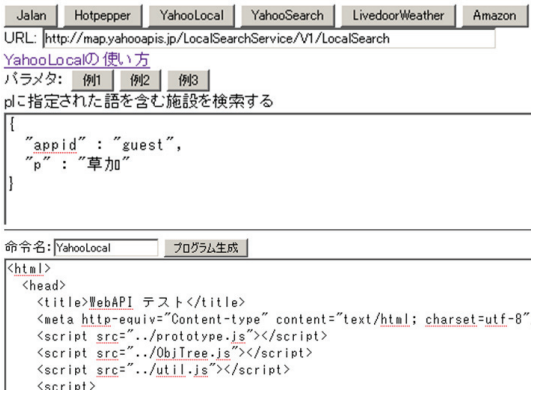

図 2 JavaScript + Web API による WBI
Fig. 2 WBI in JavaScript and Web API

2.1 様々な種類の言語で書かれたプログラムを安全に動かす

WBI では，学習者が書いたプログラムが Web サーバなどに不具合を起こす可能性を排除する必要がある．

図 1 の PHP の実習環境では，学習者の書いたプログラムを PHP のインタプリタを使ってそのまま動作させしていたので，Web サーバの重要なファイルを読み書きされる可能性や，外部のネットワークに望ましくないアクセスをされる恐れがあった．

一方，図 2 の JavaScript の実習環境は，学習者が書いたすべてのプログラムをすべてブラウザで動かしたので，Web サーバに直接不具合を与えることはなかったが，サーバサイドのプログラミングができなくなったため，十分な演習ができなかった．

これらのことから，サーバサイドで安全なプログラムを実行することが必要であり，また，学習する内容によって多くの言語に対応させる必要もあるといえる．

2.2 障害による停止を防ぐ

WBI による実習環境は，通常，教室外の Web サーバに設置することが多い．このとき，サーバの障害やネットワークの寸断があると教室から実習環境が使えなくなってしまう．

普通の実習においては，実習に用いる処理系などは教室内のローカルな端末にインストールされているので，万が一ネットワークが切れても，プログラムを実行するこができる．しかし，WBI では実習そのものを Web で行なっているので，ネットワークが切れたら実習ができなくなる．このためネットワークやサーバの障害は致命的となる．このため，ネットワークがつながっていなくても実習が続けられる環境が必要になる．

2.3 学習結果を統合的に管理する

授業の進行中や授業終了後に，各学習者の学習結果（学習者が作ったプログラム，レポート等）の解析をする場合がある．学習者によっては，複数の WBI を用いている場合もあるので，学習結果を横断的に解析するにはすべての WBI のデータが統合的に管理できる必要がある．

3. soyText の特徴

前節で示した要件を解決するため，Web フレームワーク soyText を開発した．soyText は次のような特徴をもつ．

- **サンドボックス環境でのスクリプト言語の環境を提供する**
  soyText のシステムには，各種スクリプト言語のスクリプトエンジンを搭載し，学習者が書いたプログラムはすべてスクリプトエンジンを通じて実行させる．これにより，学生が不用意に書いた危険な動作を抑制させるようにする．また，多数の言語に対応することによって，単一のシステムで様々な授業の WBI を作成できるため，学習結果の統合管理が容易になる．
- **サーバで動いている環境をそのままダウンロードして，各ユーザの手元の PC でも Web サーバとして動作する**
  ネットワークのない環境で学習する場合や，ネットワーク・サーバが止まっている状態においても，予め手元にダウンロードしておいたサーバの環境をダウンロードしておくことで，手元の PC でも同じように学習ができる．また，ダウンロードした環境は，特別なインストールや設定の作業なしに簡単に起動できるようにしておく．
- **手元の PC のプログラム・データとサーバのプログラム・データの同期ができる**
  前項のように，手元の環境で学習を行った場合，あとでサーバと接続できる状態になったときに，手元の環境とサーバの環境を同期させ，学習結果をサーバに反映させることができる．また，サーバのプログラムやデータに変更があった場合は，手元の環境にそのプログラムやデータを反映させることもできる．

soyText の利用形態の例を**図 3** に示す．

- 通常時（図 3 の左側）は，ネットワーク上のサーバにインストールされている soyText のシステム（以下，システム）だけが稼働しており，ユーザは，個人がもつ PC から Web ブラウザを通じて

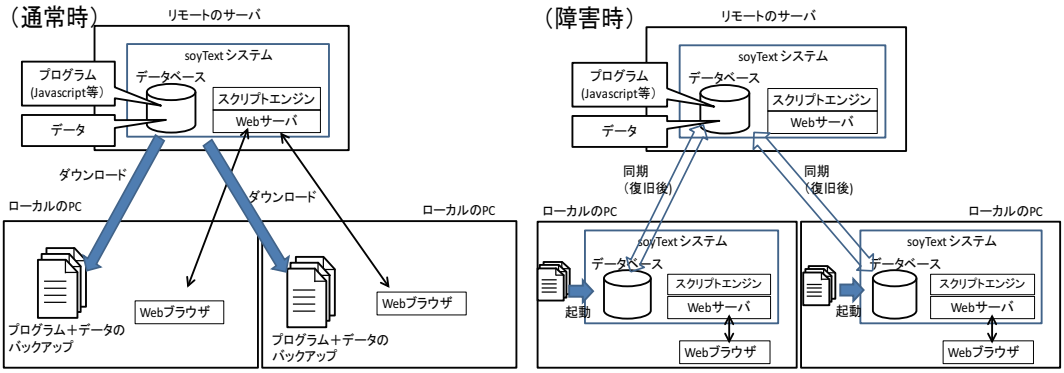

図 3 soyText の利用形態
Fig. 3 Assumed usage in soyText

システムにアクセスする．

- 定期的に（毎回の授業の終わりなど）に，システムからプログラムおよびデータをダウンロードさせ，各自のローカルな PC にバックアップさせておく．
- 万が一，ネットワークやサーバの障害があった場合（図 3 の右側）は，手元にバックアップしておいたシステムをローカルの PC で起動し，Web ブラウザから localhost にアクセスすることで，システムの利用を継続することができる．
- 障害が復旧した後，サーバにあるプログラムおよびデータと同期をすることによって，サーバにも障害中に変更したデータを反映させることができる．

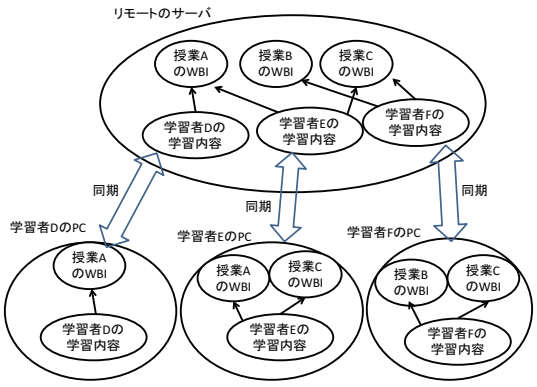

図 4 授業における soyText のデータ配置例
Fig. 4 Data deployment of soyText in classrooms.

このような形態にすることによって，Web アプリケーションとしての手軽さ（Web ブラウザからアクセスできる）を損なうことなく，ネットワークが不通の状態でも最低限の作業ができるようになる．

なお，Web サーバにインストールされたシステムには，すべての授業で用いる WBI のプログラムおよびすべての学習者の学習結果を格納しておくが，各個人の PC にダウンロードされるデータは，**図 4** で示しているように，各ユーザの手元の PC には，各自が必要とするプログラムおよびデータだけに限定することができる．これにより，個人のホームディレクトリのような，容量に制限のある環境でも最低限の容量で動かすことができる．

4. 実　　装

4.1 実装の方針

前節で述べたように，soyText は各学習者の手元の環境でも動作するような形態になっている．学習者の手元の環境は，教室の PC 端末など，インストールに制限のある環境である場合が多く，スクリプトの処理系，Web サーバ，データベースのシステムを一式インストールするのは大変であるため，次のような方針で実装を行った．

- **Java を用いて実装する**
  Java は，数多くのプラットフォームで動かすことができる．アプリケーションは実行可能な jar 形式にまとめて配布できるため起動が簡単に行える．また，Rhino[3](JavaScript)，JRuby[4] (Ruby)，Jython[5](Python)，Quercus[6](PHP) など，多くの言語のスクリプトエンジンが実装されている．
- **サーバやデータベースを組み込んで実装する**
  教室などの授業環境では，インストールできるソフトウェアが限定されているため，DBMS やサーブレットコンテナを別途インストールすることは難しい．
  そこで，データベースには Sqlite の Java 実装である SqlJET[7]，Web サーバは，スタンドアロンな Web サーバである NanoHTTPD[8] を用いた．

- **プログラムやデータ間の参照関係を保障する**
  3 節の最後で述べたように，同期を行う際に，手元の環境は最低限必要なデータをダウンロードすることで，必要な容量を少なくすることができる．このとき，あるプログラムやデータを正しく利用するために必要な他のプログラムやデータ（これを,「参照している」プログラムやデータと呼ぶ）が何であるかを，フレームワークである soyText 自身が把握しておく必要がある．そこで，soyText では，すべてのプログラムやデータに，それぞれが参照しているプログラムやデータの情報を付加するようにしている．

4.2 データベースの構造

soyText においては，サーバ・クライアント間でプログラムとデータがすべて同期される．同期の仕組みを単純に実装するため，プログラムとデータを同一のものとして扱えるようにした．プログラムとデータをあわせて「文書」と呼ぶ．

データベースは「文書テーブル」というテーブルをもつ．文書テーブルには複数の文書がレコードとし て入っている．各文書は，各種スクリプト言語のオブジェクトとして表現される．

文書は次の属性をもつ：

- **id**
  この文書の ID をあらわす．他のシステムの間でも ID は一意である必要がある． 現実装では「各システムが生成した通し番号@システム ID」という形式である．システム ID は，各システムに振られた ID であり，この文書を最初に作成したシステムの ID となる．同期を行うことにより，他のシステムが作成した文書の複製が作られることがある．したがって，他のシステムの ID をもった文書が格納されることもある．
- **lastUpdate**
  この文書が最後に作成または変更された日時をあらわす．以前に同期したことがあるシステムと再度同期する際には，この日時を用いて変更があった分だけを同期する．
- **content**
  この文書があらわすプログラムまたはデータをスクリプト言語で記述する．記述の内容によっては，文書が他の文書を参照することもある．記述例は 5 に後述する．
- **language**
  content を記述するスクリプト言語を表す．現在は”JavaScript”のみが対応している．

4.3 同　　期

他の soyText システムとデータベースの同期を行うことによって，文書の内容をそれぞれ同じ状態にすることができる．データベースにはプログラムとデータが両方含まれるため，プログラムの更新も含めてアプリケーションを同じ状態にすることができる．

同期には次の 2 つの方式がある．

- **全同期**
  同期元のシステムと，同期先のシステムがお互いのすべての文書を保持する方式である．最初に同期する場合は，同期先のすべての文書がダウンロードされ，同期元のシステムがもつすべての文書をアップロードする．2 回目移行の同期においては，最後の同期以来更新のあった文書だけをダウンロード・アップロードする．
- **部分同期**
  特定の条件に一致する文書と，それらの文書から（再帰的に）参照される文書だけをダウンロード・アップロードする方式である．条件は同期元と同期先で異なっていてもよい．例えば，ローカルのシステムから，リモートのシステムへは，すべての文書をアップロードするが，リモートからローカルへは，ローカルのユーザと関係する文書（そのユーザが作った文書など）だけをダウンロードするように設定することも可能である．

5. 文 書 の 例

ここでは，データベースに格納されるデータ・プログラムの例を示す．なお，soyText には標準の文書エディタが組み込まれており，**図 5** のように Web ブラウザから編集可能である．

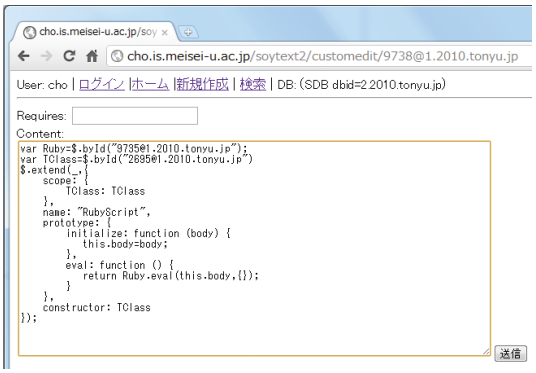

図 5 soyText 標準の文書エディタ
Fig. 5 Default editor embedded in soyText

5.1 単純な文書

図 6 に，単純な文字列属性をもつ JavaScript オブジェクトを，文書に記述した例を示す．

```
id: 1@2011.foo.com
content:
  $.extend(_,{
    name: "hello.txt",
    body: "Hello, world"
  });
```

図 6 単純な文書
Fig. 6 a simple document

変数$および 変数_ は soyText から自動的に渡される．変数 $ はデータの構築を手助けするオブジェクトを示している．変数 _ には文書をあらわすオブジェクト（文書オブジェクト) が初期化されていない状態で入っており，`$.extend` メソッドを通じて _ にデータを書き込んでいる．

5.2 他の文書を参照する文書

図 7 に，他の文書を参照する文書の例を示す．他の文書を参照する場合は，`$.byId` メソッドを用いて ID から文書オブジェクトを取得し，文書オブジェクトに書きこむ．content 内部で宣言されている変数とその値は，文書オブジェクトの”scope” というプロパティに書きこまれる．これは，プログラムからのデータの書き換え (後述) を行うときに，宣言されていた変数の一覧を文書オブジェクトに書き戻すときに用いられる．

```
id: 2@2011.foo.com
content:
  var hello=$.byId("1@2011.foo.com");
  $.extend(_,{
    scope: {hello: hello},
    name: "my folder",
    hello: hello
  });
```

図 7 他の文書を参照する文書
Fig. 7 a document which refers other document

5.3 プログラム（関数）を含む文書

文書のデータには関数を含むことができる．**図 8** に，関数を含む文書の例を示す．

図 9 のように，外部のデータを参照する関数も作れる．

5.4 Web アプリケーション文書

doGet というプロパティに関数を設定した文書を Web アプリケーション文書と呼ぶ．この文書に Web ブラウザからアクセスすると，doGet に設定した関数

```
id: 3@2011.foo.com
content:
  $.extend(_,{
    name: "displayResult",
    display: function (s) {
      return "Result :"+s;
    }
  });
```

図 8 関数を含む文書
Fig. 8 a document having a function

```
id: 4@2011.foo.com
content:
  var hello=$.byId("1@2011.foo.com");
  $.extend(_,{
    scope: {hello: hello},
    name: "displayHello",
    display: function () {
      return "Result :"+hello.body;
    }
  });
```

図 9 外部の文書を関数から参照する
Fig. 9 a function refers other document

が呼ばれて，Web ページが動的に生成される．**図 10** に，Web アプリケーション文書の例を示す．doGet は，Servlet における request と response オブジェクトを受け取り，ページの生成を行う．

```
id: 5@2011.foo.com
content:
  $.extend(_,{
     name: "changeHello",
     doGet: function (request,response) {
        var w=response.getWriter();
        w.println("Hello, world");
     }
  });
```

図 10 Web アプリケーション文書
Fig. 10 a Web application document

5.5 プログラムからのデータの書き換え

図 11 に記述されたページを実行すると，実行後，1@2011.foo.com の文書の content が**図 12** のように自動的に書き換わる．この例では，文字列の値を書き込んでいるが，他のオブジェクトへの参照や，配列，ハッシュ，関数などを書き込むことも可能である．

5.6 検　　索

図 13 は，データベース内の文書の検索をするプログラムの例である．文書を検索するには，db という変数から参照される組み込みオブジェクト（db オブジェクト）を用いる．ここでは，検索条件として name というプロパティの値に hello という文字列を含む文書

```
id: 6@2011.foo.com
content:
  var hello=$.byId("1@2011.foo.com");
  $.extend(_,{
    name: "world2japan",
    doGet: function (request,response) {
      hello.body="Hello, Japan";
      hello.save();
      w.println("hello.body="+hello.body);
    }
  });
```

図 11 文書を書き換えるプログラム
Fig. 11 a program modifying other document

```
id:
  1@2011.foo.com
content:
  $.extend(_,{
    name: "hello.txt",
    body: "Hello, Japan"
  });
```

図 12 書き換え後の内容
Fig. 12 changed content after modification

を指定している．条件に一致したすべての文書について，each メソッドに指定した関数を呼び出して処理を行う．

```
id: 8@2011.foo.com
content:
  $.extend(_,{
    name: "changeHello",
    doGet: function (request,response) {
      var w=response.getWriter();
      db.q("name","hello").each(function (doc) {
        w.println("Found :<br>");
        w.println("name="+doc.name+"<br>");
        w.println("body="+doc.body+"<br>");
      });
    }
  });
```

図 13 文書の検索
Fig. 13 searching documents

5.7 オブジェクト指向

JavaScript では，関数オブジェクトを用いてオブジェクトのプロトタイプを定義する．ここでは，プロトタイプを定義するための関数オブジェクトを便宜上「クラス」と呼ぶ．

soyText の文書は，すべて関数オブジェクトとして見なすことができるので，任意の文書の prototype プロパティに必要なメソッドを定義することで，クラスが定義できる．

図 14 は，クラスを定義した例である．ここでは，Ruby のスクリプトをあらわすクラスを定義している．initialize メソッドはコンストラクタとして定義され，引数に Ruby のプログラムを表す文字列を受け取って初期化する．

```
id: 9@2011.foo.com
content:
  var Ruby=$.byId("xxx@2011.foo.com");
  $.extend(_,{
     name: "RubyScript",
     prototype: {
        initialize: function (body) {
           this.body=body;
        },
        eval: function () {
           return Ruby.eval(this.body);
        },
        doGet: function (request,response) {
          var w=response.getWriter();
          w.println(this.eval());
        }
     }
  });
```

図 14 クラスをあらわす文書
Fig. 14 document represent a class

図 15 に示した文書は，図 14 のクラスのインスタンスである．constructor プロパティに，この文書のクラスとなる文書を指定している．この文書は，図 14 で指定された doGet 関数をもっているため，Web アプリケーション文書として実行可能である．実行すると，body プロパティに書かれている Ruby のスクリプトの実行結果 Web ブラウザに表示する．

なお，図 15 に書かれた Ruby のプログラムは，Javascript の文字列リテラルで記述されているため，図 5 のような標準のエディタで編集するのは難しいため，自分でエディタを作成して編集方法をカスタマイズすることもできる．

6. 実　　例

筆者が担当しているプログラミングの授業においては soyText を用いて授業支援の Web アプリケーションを開発し，授業で学生に使わせている．

ここでは，プログラミング言語「ドリトル[9]」を用いた Web アプリケーション開発環境「D-rails[10]」の例を示す．D-rails の動作画面を**図 16** に示す．この画面は Web ブラウザ上で動作し，Web ブラウザからプログラム（ページ）を記述し，表示・実行を行うことができる．D-rails によるプログラムの例を示す．**図 17** は，データベースを新しく作成する例であり，**図 18** および**図 19** は，データベースのデータを表示する例

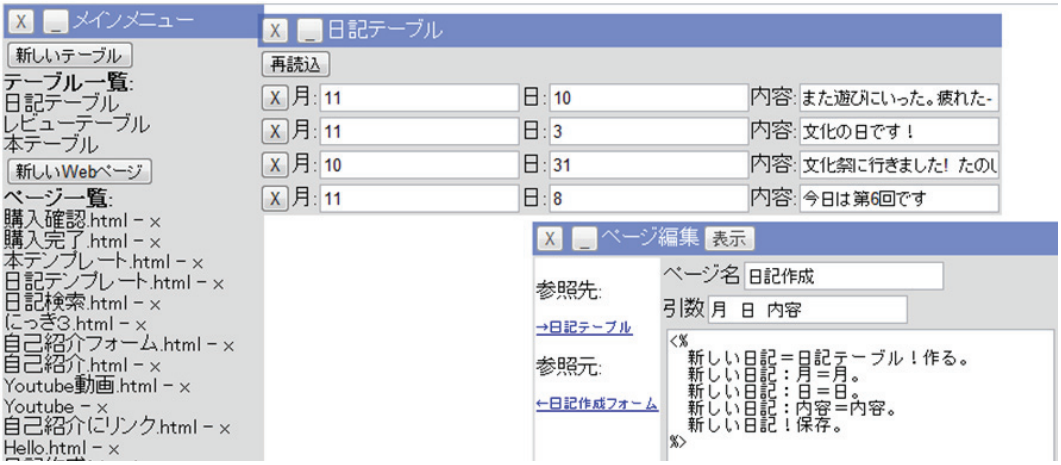

図 16 D-rails の動作画面
Fig. 16 Screenshot of D-rails

```
id: 10@2011.foo.com
content:
  var RubyScript=$.byId("9@2011.foo.com");
  $.extend(_,{
     constructor: RubyScript,
     name: "helloRuby",
     body: "class Test \n"+
     "    def hello(s)\n"+
     "       \"Hello, #{s} world\"\n"+
     "    end\n"+
     "end\n"+
     "Test.new.hello(\"soyText\")"
      /*
      #body の内容は次の通り：
      class Test
        def hello(s)
          "Hello, #{s} world"
        end
      end
      Test.new.hello("soyText")
      */
  });
```

図 15 クラスのインスタンス
Fig. 15 an instance of class

である．

ドリトルは，Java で実装されており，soyText のスクリプトエンジンとして取り込むことが可能である．

```
<html><body>
<%  新しい日記＝日記テーブル！作る.
    新しい日記：月＝"10".    新しい日記：日＝"12".
    新しい日記：内容＝"大学で授業を受けました".
    新しい日記！保存.  %>
</body></html>
```

図 17 D-rails のレコードの作成
Fig. 17 Creating a database record in D-rails

```
名前：日記検索.html
引数：(なし)

<h1>日記を表示する</h1>
<% 日記テーブル！ 検索 (日記テンプレート) 実行. %>
```

図 18 D-rails のレコード表示
Fig. 18 Searching record in D-rails

```
名前：日記テンプレート.html
引数：日記

<h2><% 日記：月  %> 月  <%日記：日 %> 日の日記</h2>
<% 日記：内容 %><br>
```

図 19 レコードの表示に用いるテンプレート
Fig. 19 A template used for displaying records

7. 今後の展開

7.1 JavaScript 以外の言語を用いた文書の表現

現状，文書の記述は contents 属性に JavaScript のプログラムを書くことで実現している．それ以外の言語で書かれたプログラムは，図 15 のように JavaScript のオブジェクトの中に文字列データとして格納する必要がある．今後は，Ruby などの他の言語で書かれたオブジェクトを直接文書に格納できるように改良していく．

ただし，JavaScript は関数の内容を文字列に変換することが容易であるため，関数を含んだオブジェクトを，5.5 項のように他のプログラムから書き換えても関数の内容が保持される．一方，他の言語においては，

関数を文字列に変換する機能をもたないものが多いため，そのような機能をスクリプトエンジンに追加する必要も出てくる．

7.2 Java の依存度を下げる

現状の soyText の実装では，スクリプトエンジンや Web サーバの他にも，図 5 のような Web インタフェースや，検索の仕組みなどを Java で実装している．これらを JavaScript などのスクリプト言語に置き換えていくことで，最終的にはシステム全体をスクリプト言語だけで実装することを目指している．これにより soyText を Java 以外の言語で実装されたスクリプトエンジンを使って実装することも可能になり，Java のない環境でも動かせるようになる．

8. ま と め

本発表では，Web フレームワーク”soyText” を提案した．soyText では，データとプログラムを一体化し，Java で動作するスクリプト言語によって書かれたプログラムを動作させることで，多くの環境で簡単に動作させることができる．また，他の soyText 環境と同期することによって，場所を選ばず環境を構築することができる．

参 考 文 献

1) 兼宗進, 長慎也: 文科系大学におけるサーバーサイドプログラミング授業の試み, 情報処理学会研究報告，2006-CE-83, pp. 141–148 (2006).
2) 長慎也: 文科系大学における Web サービス構築実習, 情報処理学会研究報告，2008-CE-97, pp. 61–68 (2008).
3) foundation, M.: Rhino - Javascript for Java http://www.mozilla.org/rhino/.
4) jruby.org: JRuby http://jruby.org/.
5) jython.org: Jython: Python for the Java Platform http://www.jython.org/.
6) resin: Quercus http://quercus.caucho.com/.
7) Software, T.: SQLJet :: Pure Java SQLite http://sqljet.com/.
8) Elonen, J.: NanoHTTPD http://elonen.iki.fi/code/nanohttpd/.
9) 兼宗進, 久野靖: プログラミング言語「ドリトル」, http://dolittle.eplang.jp/.
10) 長慎也: D-rails - Web アプリケーション学習用フレームワーク, 情報処理学会研究報告，2011-CE-108, pp. 1–12 (2011).