卒業論文

Ruby on Railsを用いたMaple演習書の作成システム

関西学院大学 理工学部 情報科学科

2717 谷垣 雄一

2008年3月

指導教員　西谷 滋人 教授

概要

Mapleは，数式処理，数値計算，グラフ作成などを行うソフトウェアのひとつである．他にも制御設計，回路設計，光学設計など，様々な分野の研究開発に利用されている．しかし，日本語の標準的な教科書は見当たらない．米国では数多くの教科書が存在しているが，数学における学習項目の順序，レベルが日米では全く異なるため，米国の教科書を日本で翻訳して使うことは適切ではない．そこで本研究では，日本人がイメージしやすいチャート式のスタイルを雛形とし，個人的なMapleの演習書を容易に作成するシステムの構築を行う．

ここでシステムの構造として，通信販売サイトであるAmazonのような構造を考える．Amzonは，多くの商品の中からユーザーが好きな商品をカートに入れ，レジで会計を済ませた後，届くようになっている．この商品をチャート式の項目に置き換え，各項目をpdfファイルにしアップロードし，必要な項目をカートに入れ，順序を決めた後，アップロードしてあるpdfファイルを 1 つのpdfファイルにまとめてダウンロードし，演習書を作るという構造を考えた．

開発プラットフォームには，Rubyを使用したWebアプリケーションフレームワークのRuby on Railsを用いた． 通信販売サイトは，ブログなどと共に代表される典型的なWeb2.0のコンテンツである．Web2.0とは，従来の静的で一方通行的なWebコンテンツとは違い，誰でもウェブを通して情報を発信できる双方向のWebコンテンツで，更新が容易である．Ruby on Railsでは，少ないコードで開発でき，通常 1 画面ごとに手作業で用意していたスケルトンがスクリプト 1 つでできるため，Web2.0のコンテンツが簡単に作れる．Ruby on Railsのスクリプトによって作られた，雛型にある新規登録，詳細表示，編集，削除に加え，必要な機能を追加していく．

これに必要な機能として，ファイルのアップロード，ショッピングカート，並べ替え，ファイルの統合が挙げられる． ファイルのアップロードは，アップロードしたファイルデータをデータベースに格納するのではなく，ローカルファイルとして保存することにした．こうすることによって，ファイル統合のときに添付されたファイルを呼び出しやすくした． ショッピングカートは，公開されているコードを利用し，それを適合させることで実装した．カートは，商品をキーとし個数を値とするハッシュとして表現できるが，システム上，個数は必要ないと判断し，この値を順序にすることで並べ替えをできるようにした．ファイルの統合は，コマンドライン上でpdfファイルを連結させることができるpdftkというツールと，Rubyのシェルスクリプトとしての機能を利用した．

第 1 章　緒言

1.1　Mapleのテキスト

1.1.1　Maple

Mapleは1980年代前半にカナダのウォータールー大学で開発された微分方程式，代数計算，データ解析などの手計算では困難な問題を，簡単なプログラムを入力して実行するだけで，解答を出力するソフトウェアである．また数値を出力するだけでなく，グラフ描画機能にも優れていて，３次元のグラフィックスやアニメーションの作成が可能である．他にも制御設計，回路設計，光学設計など様々な分野の研究開発に利用されている．そして最近では，小学校，中学校，高校などの初等教育の現場における数学，理科の授業から，大学や企業の研究機関に至るまで幅広いユーザー層が開拓されつつある．

しかし，マニュアルや解説書の不足は深刻で，全く使い方を知らない人間，あるいは周囲に使い方を知る人材がいない人間のおかれた状況は悪い．サードパーティによるマニュアル本の出版がされておらず，新しいバージョンに関しては，索引のない付属の解説書に頼らざるを得ないためである．さらに，一部のコマンドを除き，ヘルプ内の文章をコピーアンドペーストによって他の箇所に保存することが出来ない．このことは自分用のマニュアルを作ったり教材を作ったりするうえで致命的と言えるほど不便である．

1.1.2　チャート式

チャート式は，1926年に「チャート式数学」が発行されて以来、日本では，学習参考書の草分け的存在として知られている．チャート式は「１ページに単元をおさめ，その単元が細分化されている」，「解答付きの例題と演習などで構成されている」といった特徴を持っている．これならWebを使ったテキストでも表現できると考えた．そして，チャート式テキストは，学習内容の重点をおさえ，学習者自身が問題における急所の発見と，その解法の導き方を示すことを主眼としている．これはMapleのテキストにも必要な考え方である．

また，チャート式は，現在，中学・高校の数科目のものが発売されているが，とりわけ高校数学のものが広く一般的に知られており，「チャート」と言えば高校数学のチャート式を指す場合が多く，誰でも高校数学を学習したときに一度は見たことがあるというほどでメジャーある．よって日本人にとってチャート式は，テキストとして容易にイメージを膨らませる格好のスタイルであると言える．

1.2　研究目的

最近では，インターネット上に多くのテキストが掲載されている．Mapleに関してもインターネット上には，中学，高校，大学の教師などが自分の講義用に作ったノートが複数公開されている．しかし，そのようなテキストをダウンロードして利用しようとすると，

余分な項目が入っていたり，必要な項目が 1 カ所にまとまっていなかったりすることがよくある．そのため，個人的に必要なテキストのみを 1 つのファイルにするのは困難である．チャート式Mapleは，見開き 2 ページの項目，Table状にまとめられている，解法と課題で提供されるという特徴を持っている．昨年，菅野氏は，このチャート式Mapleから個人の好みに合った演習書作成システムのWikiによる実装を試みたが，困難が生じた．つまり，想定している機能はWikiのプラグインとして作る必要があった．

そこで本研究では，より基礎的なフレームワークを用いた実装を目指し，Mapleの解説項目から必要な項目を選んで個人的な演習書を容易に作成できるシステムの構築を目的とする．

第2章　開発プラットフォーム

2.1　Ruby on Rails

2.1.1　Rubyとは

Rubyは，まつもとゆきひろ氏により開発され，1995年12月にfj上で発表されたオブジェクト指向スクリプト言語である．機能として，クラス定義，ガベージコレクション，強力な正規表現処理，マルチスレッド，例外処理，イテレータ・クロージャ，Mixin，演算子オーバーロードなどがある．．Rubyにおいては整数や文字列なども含めデータ型はすべてがオブジェクトであり，純粋なオブジェクト指向言語といえる．

2.1.2　Ruby on Railsとは

Ruby on RailsはデンマークのDavid Heinemeier Hanssonにより作成されたフレームワークで，2004年7月に公開され，2005年12月13日にバージョン1.0がリリースされた．Ruby on Railsは，フレームワークとしては一般的なMCV型に属する．MCV型とは，Model-View-Controllerの頭文字をとった言葉で，データとそれに関わる処理を担う「モデル(Model)」，表示・出力を行う「ビュー(View)」，これらのビューとモデルを制御する「コントローラー(Controller)」といった３つのコンポーネントを基礎とするアーキテクチャである．このように機能を分割しておくと，ビュー・コントローラーとモデルが疎結合になることによってモデルの再利用性が高まり同時にビューの入れ替えが容易になり，ソースコードの見通しもよくなる．

Ruby on Railsには，基本理念として「同じことを繰り返さない」(DRY:Don't Repeat Yourself)と「設定よりも規約」(CoC:Convention over Configuration)がある．「同じことを繰り返さない」(DRY)というのは，Andrew Hunt氏とDevid Thomas氏が打ち出した原則で，定義などの作業は一回だけですませ，冗長なだけでなく往々にして間違いの元になる，重複や繰り返しを避けろという意味である．「設定よりも規約」(CoC)とは，Ruby on Rails発祥の哲学で，レアなケースをも想定して冗長になりがちな設定を極力排除し，規約に従うことでお決まりの動作を実現させようという思想である．

また，Ruby on Railsはフレームワークであるだけでなく開発環境でもある．Ruby on Railsには，コード生成機能があり，基本理念の１つである「設定よりも規約」に基づくテンプレートが配置され，通常１画面ごとに手作業で用意していたスケルトンがスクリプト１つで生成でき，また，データベースレコードの新規作成，参照，更新，削除を行うような単純なWebアプリケーションもまたスクリプト１つで生成できる．さらに，Ruby on Railsに用意されているのはコード生成機能だけではなく，すぐに動かして試せる簡易Webサーバやインタラクティブシェルといったスクリプト，コード生成時にはテストコードのひな型も同時に生成さる．これらにより，Webアプリケーションの開発が容易になる．

2.1.3　なぜRuby on Railsを利用したか

本研究で構築しようとするシステムの構造として，表 1 のように典型的なWeb2.0のコンテンツである通信販売サイトのAmazonと対応させて考える.

表2.1 ：Amazonとチャート式の対応.

Amazon	チャート式
商品	課題　章
カート	選ぶ　抜き出す
買う	ダウンロード
届く	演習書を作る

Amazonで買い物をするような感覚で項目を選択し，選択された項目を並べ替えた後に 1 つのファイルとして出力するシステムにする．また，ジャンルと難易度を登録しておき，項目のタイトルも含めて検索できるようにしたり，チャートに新規の項目を追加する際に自動的にジャンルと難易度に対応した場所に配置するようにする.

このようなシステムには，容易にWebアプリケーションが作成できるRuby on Railsが最適と考えた.

第3章　準備

3.1　Ruby on Railsのインストール

3.1.1　RubyGemsのインストール

RubyGemsは，Rubyにおけるパッケージマネージャであり，Ruby on RailsはRubyGemsを使ってインストールするのが一般的なため，ここではRubyGemsのインストールを解説する．RubyGemsは以下のWebサイトからダウンロードできる．

RubyGems
http://docs.rubygems.org/

RubyGemsのダウンロード
http://rubyforge.org/frs/?group_id=26

ダウンロードするファイルは，tgzファイルおよびzipファイルであり，ここではrubygems-0.9.2.tgzをダウンロードする．
それを以下のように展開して，そのディレクトリに移動してインストールする．

コマンド
```
$ tar -xvf rubygems-0.9.2.tgz
$ cd rubygems-0.9.2
$ su
password: ――――――――　スーパーユーザーのパスワードを入力
# ruby setup.rb config
# ruby setup.rb setup
# ruby setup.rb install
# exit
```

exitでスーパーユーザーモードを終了してRubyGemsのインストールは完了．
suコマンドでrootになれない場合は，sudoコマンドを利用する．前記のように#から始まるコマンドは，sudoを前に付けて実行する．

3.1.2　Ruby on Railsのインストール

以下のように，su(あるいはsudo)コマンドでスーパーユーザーになって実行する．

コマンド
```
$ su
password: ――――――――　スーパーユーザーのパスワードを入力
# gem install rails -y
# exit
```

exitでスーパーユーザーモードを終了してRuby on Railsのインストールは完了.

3.2　MySQLのインストール

MySQLのインストールは，MySQLのWebサイトからダウンロードして行うのが一般的.

MySQL AB::MySQL Downloads
http://dev.mysql.com/downloads/index.html

MySQLのパッケージには,
・mysql-standard-5.0.21-osx10.4-i686.pkg …… MySQL本体のインストーラ
・MySQL.prefPané ……MySQLのシステム環境設定パネル
・MySQLStartupItem.pkg ……MySQLを起動時のサービスに組み込む
・ReadMe.txt
の４つのファイルがある.
まず，MySQL本体をインストーラを用いてインストールし，次にMySQL.prefPanéを実行すると，インストールするかどうか尋ねられるので，「はい」と答えて，インストールする．そして， MySQLStartupItem.pkgをインストールすると， 起動時のサービスに組み込まれる．なお，MySQLの起動と終了は，これにより，システム環境設定で行える.
このとき，MySQL自体は，usr/local/mysql/
にあり，mysqlやmysqladminコマンドなどは， usr/local/mysql/binディレクトリに存在するが，このディレクトリにはパスが通っていないため，.profileファイルで
export PATH=$PATH:/usr/local/mysql/bin
と書き加えパスを通しておく.
また，デフォルトのエンコーディングの設定はemacsを使い用意する必要がある.

コマンド
```
$ sudo emacs /etc/my.conf
```

リスト：/etc/my.conf
```
[mysqld]
default-character-set=utf8
[mysql]
default-character-set=utf8
[mysqldump]
default-character-set=utf8
```

初期インストール時はアクセスが解放されている状態なので，インストール後にはまず，MySQLのrootユーザにパスワードを設定する.

コマンド
mysqladmin -u root password “ パスワード”

3.3　プロジェクト

3.3.1　プロジェクトの作成

まず，プロジェクトの作成の前に，プロジェクトを入れておくディレクトリmainを用意して，そのディレクトリに移動する．その後，railsコマンドでプロジェクトを作成する．

コマンド
$ mkdir main
$ cd main
$ rails main
$ cd main

これで，ディレクトリchart以下にWebアプリケーションのひな型となる様々なファイルが作成される．以降，ファイルの位置を示すときは，このルートディレクトリを基点として解説する．

3.3.2　データベースの設定

データベースの設定は，config/database.ymlを変更して行う．

リスト：config/database.yml(コメント除く)
```
development:
  adapter: mysql
  database: main_development
  username: root
  password: パスワード
  socket: /tmp/mysql.sock
  encoding: utf8

test:
  adapter: mysql
  database: main_test
  username: root
  password:パスワード
  socket: /tmp/mysql.sock
  encoding: utf8
```

```
production:
  adapter: mysql
  database: main_production
  username: root
  password: パスワード
  socket: /tmp/mysql.sock
  encoding: utf8
```

　このように，

```
socket: /tmp/mysql.sock
encoding: utf8
```

を書き加え，password:にMySQLのパスワードを入力する．

　そして，データベースにアクセスできるようにmysqlコマンドを使い，権限を設定する．

コマンド

```
$ mysql -u root -p
パスワード
mysql> create database recipe_test;
mysql> create database recipe_deveropment;
mysql> create database recipe_production;
mysql> grant all on recipe_test.* to root@localhost
mysql> grant all on recipe_developmennt.* to root@localhost
mysql> grant all on recipe_production.* to root@localhost
mysql> ¥q
```

前半の３つでデータベースを作成し，後の３つでユーザにアクセス権を与え，最後の¥qでmysqlコマンドを終了している．

3.3.3　モデルの作成

　モデルの作成するには， script/generate modelを使う．

　script/generateはジェネレータを動作させるためのスクリプトで，script/generate modelとはmodelジェネレータを起動させるという意味になる．

コマンド

```
$ ruby script/generate model cont
```

　script/generate modelを実行すると，マイグレーションの設定を記述するためのファイルdb/migrate/00x_create_conts.rbが作成される．これを，エディタで編集し，データモデルの設定を行う．

リスト：db/migrate/00x_create_conts.rb

```
class CreateConts < ActiveRecord::Migration
  def self.up
    create_table :conts do |t|
      t.column :name, :string
      t.column :level, :string
      t.column :genre, :string
    end
  end

  def self.down
    drop_table :conts
  end
end
```

下線部のl4〜l6でカラムの型を設定している．本研究では，上からそれぞれ，課題のタイトル，難易度，ジャンルを設定している．

これが終わったら，rakeコマンドでマイグレーションを実行し，データベーステーブルを作成する．

コマンド

```
$ rake migrate
```

rakeコマンドでテストを実行し，config/datagase.ymlでのデータベース設定が正しく行われているかを確認する．

コマンド

```
$ rake
```

このテストで，

```
1 tests, 1 assertion, 0 failures, 0 errors
```

と出力されればテストは成功している．

3.3.4　コードの生成

登録画面を簡単に仕上げるために，scaffoldジェネレータを使う．scaffoldジェネレータを使うと，データベースのスキーマを基に，モデル，コントローラ，ビューを簡単に生成できる．

コマンド

```
$ ruby script/genetate scaffold Chart
```

これで，コントローラのapp/contorollers/charts_controller.rbと，ビューのapp/views/charts/_form.rhtml，～/list.rhtml，～/show.rhtml，～/new.rhtml，～edit.rhtmlなどが自動で生成される．

3.3.5　WebサーバのWEBRick起動

Ruby on RailsにはRubyによる組み込みのWebサーバWEBRickを使ってRailsのアプリケーションを実行できる．WEBRickは次のようにして起動させる．

コマンド
```
$ ruby script/server
```

その後，ブラウザからhttp://localhost:3000/chartsにアクセスすると，本研究のWebアプリケーションを実行することができる．

以降は，出力を見ながら，主にビューであるrhtmlファイルと，コントローラーのcharts_controller.rbを編集し，目的のWebアプリケーションに近づけていく．

第4章　システム

4.1　リスト (http://localhost:3000/charts/list)

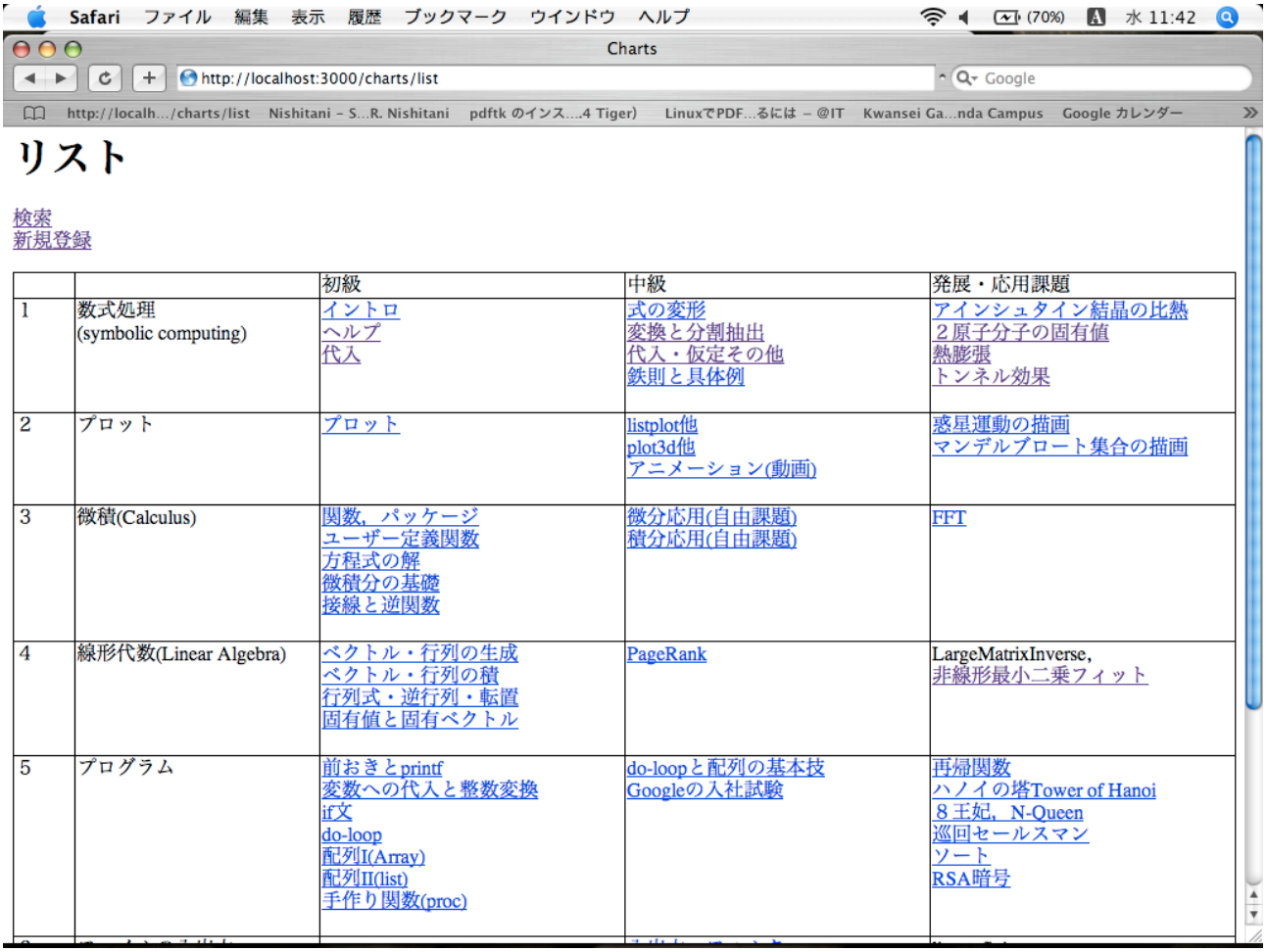

図4.1:リストのページ

http://localhost:3000/charts にアクセスすると，このリストのページに来るようになっており，「検索」をクリックすると検索のページに，「新規登録」をクリックすると新規登録のページに，項目のタイトルをクリックすると詳細のページに行くようになっている．

4.2　新規登録 (http://localhost:3000/charts/new)

図4.2:新規登録のページ

ここでは，新しい項目の追加を行える．Name，Level，Genreにそれぞれタイトル，難易度，ジャンルを入力し，fileの下の「ファイルを選択」をクリックし，アップロードするファイルを選択する．最後にCreateをクリックすると，新しい項目が追加されリストのページに戻る．backをクリックすると項目を追加せず，リストの戻る．

4.3　検索 (http://localhost:3000/charts/search)

図4.3:検索のページ

「検索ワード：」の右の入力フィールドにテキストを入れ「search」を押すと，上のように検索ワードを含むタイトルをもつ項目が列挙される．これらのタイトルをクリックすると，リストのときと同様に，その項目の詳細ページに行くことができる．

4.4　詳細 (http://localhost:3000/charts/show/'id')

図4.4:詳細のページ

これは，項目の詳細ページで，アドレスの'id'には登録時に自動的に付けられる数の idが入る．項目のタイトル，難易度，ジャンルが確認でき，Fileの下の「〜.pdf」をクリックすれば，内容も見ることができる．「カートに入れる」をクリックするとこの項目がカートに入れることができる．「カートを確認する」でカートの確認ページへ，「他にも選ぶ」でリストに，「編集」でこの項目の編集ページに移れる．

4.5　編集 (http://localhost:3000/charts/edit/'id')

図4.5:編集のページ

新規登録と同じような感覚で，タイトル，難易度，ジャンルを編集することができる．「Edit」を押すと編集内容が保存され，「Destroy」をクリックするとその項目がデータベースから削除され，どちらともリストに戻る．「Back」をクリックするとその項目の詳細ページに戻る．

4.6　カートの確認 (http://localhost:3000/charts/cart)

図4.6:カートの確認のページ

今までにカートに入れた項目を確認できると同時にファイルをまとめる際の順序を決定できる．始めは上から順に並んでいるが，順序の下にあるボックスに数字を入れ「順序を更新する」を押すと，その変更が保持され，「まとめる」を押すとダウンロードのページへ移る．また，右の「取消」をクリックするとその項目はカートから削除され，「back」でリストに戻れる．

4.7　ダウンロード (http://localhost:3000/charts/result)

図4.7:ダウンロードのページ

「merge.pdf」をクリックすると，選んだ項目がまとめられたファイルが開かれ，デスクトップに保存される．

第5章　コード

5.1　リスト

リストのページには，コントローラー内のメソッド listとビューの list.rhtmlが関わっている．

リスト：app/controllers/charts_controller.rb(一部)
```
def list
    l = ['初級', '中級', '発展応用課題']
    g = ['数式処理', 'プロット', '微積(Calculus) ', '線形代数(Linear Algebra) ', 'プログラム', 'ファイルの入出力', '数値計算', '微分方程式その他のパッケージ']
   @cont11 = Cont.find(:all, :conditions => ['level like ? and genre like ?', l[0], g[0] ])
   @cont12 = Cont.find(:all, :conditions => ['level like ? and genre like ?', l[1], g[0] ]).
<略>
      end
```

リスト：app/views/charts/list.rhtml
```
<h1>リスト</h1>

<%= link_to '検索', :action => 'search' %><br>
<%= link_to '新規登録', :action => 'new' %><br>

<略>
<td
style="border-right:solid;border-left:solid;border-top:solid;border-bottom:none;border-color:black;border-width:1;" valign=top colspan=1 rowspan=1><p align="left"><font color="#000000" size="3">
                <% for cont in @cont11 %>
                <%= link_to cont.name, :action => 'show', :id => cont %><br>
                <% end %>
                </Font> </p>
              </td>
<以下略>
```

@cont11 = Cont.find(:all, :conditions => ['level like ? and genre like ?', l[0], g[0]])では，データベースの中から levelが l[0]と， genreが g[0]と一致するContを探し，その全てを@cont11に格納している．これ以下は，配列 l[], g[]に対して繰り返している．

<%= link_to '〜〜', :action => 'メソッド名' %>は「〜〜」をクリックすると，コントローラーにあるメソッドが呼び出され，ここではそのページにジャンプする．

<% for cont in @cont11 %>
<%= link_to cont.name, :action => 'show', :id => cont %>
<% end %>
これは，@cont11の各要素に対して，nameを出力し，さらにその要素の詳細ページであるshow(http://localhost:3000/charts/show/'id')にリンクさせている．

5.2　新規登録

新規登録には，メソッドの new，create，set_file_name，save_fileとnew.rhtmlに加え，テンプレートであるpartialファイルの_form.rhtmlが関わっている．

リスト：app/controllers/charts_controller.rb(一部)

```
 def new
 @cont = Cont.new
 end

 def create
   set_file_name
   @cont = Cont.new(params[:cont])
   if @cont.save
     save_file
     flash[:notice] = 'Content was successfully created.'
     redirect_to(:action => 'list')
   else
     render(:action => 'new')
   end
 end

 private
   def set_file_name
     @file = params[:cont][:file_name]
     params[:cont][:file_name] = @file.original_filename
   end

   def save_file
     if @file
       @file.binmode
       File.open("pdf/#{@cont.id}", "w") do |f|
         f.binmode
         f.write(@file.read)
       end
```

```
    end
  end
```

リスト：app/views/charts/new.rhtml

```
<h1>新規登録</h1>
<hr />

<%= start_form_tag({:action => 'create'}, :multipart => true) %>
  <%= render (:partial => 'form') %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>

<%= link_to 'Back', :action => 'list' %>
```

リスト：app/views/charts/_form.rhtml

```
<%= error_messages_for 'cont' %>

<!--[form:cont]-->
<p><label for="cont_name">Name</label><br/>
<%= text_field 'cont', 'name'  %></p>

<p><label for="cont_level">Level</label><br/>
<%= text_field 'cont', 'level'  %></p>

<p><label for="cont_genre">Genre</label><br/>
<%= text_field 'cont', 'genre'  %></p>

<p><label>file</label><br/>
<% if @cont.attribute_present?('file_name') %>
  <%= link_to("[#{h(@cont.file_name)}]", :action => 'view_file', :id => @cont) %>
<% end %>
<%= file_field 'cont', 'file_name' %></p>

<!--[Eoform:cont]-->
```

まず，アクセスされると，メソッドnewとnew.rhtmlが参照されるが，

```
 <%= render (:partial => 'form') %>
```

で，_form.rhtmlが呼ばれ，これが入力フィールドをつくる．その後，Createのボタンが押されると，

```
 <%= start_form_tag({:action => 'create'}, :multipart => true) %>
```

で，メソッドcreateが呼び出され，これがデータベースに記憶させる．さらに，メソッド

createで呼び出されているset_file_nameはアップロードしたファイルのオリジナルの名前をcont.filenameに格納するメソッドで， save_fileはアップロードしたファイルを pdf というデリレクトリにファイル名をその項目のidに変えて保存するメソッドである．これらのようにアクションとして呼ぶことを想定していないメソッドはprivateの後に書き，アクションとして呼ばれないようにしておく．

5.3　検索

リスト：app/controllers/charts_controller.rb(一部)

```
  def search
    if request.post?
      @conts = Cont.find(:all, :conditions =>
       ['name like ?', '%' + params[:keyword] + '%'])
    else
      @conts = []
    end
  end
```

リスト：app/views/charts/search.rhtml

```
<%= form_tag %>
  検索ワード: <%= text_field_tag 'keyword', params[:keyword] %>
  <%= submit_tag 'search' %>
<%= end_form_tag %>

<table>
  <tr>
    <th>ジャンル</th>
    <th>難易度</th>
    <th>タイトル</th>
  </tr>
<% @conts.each do |cont| -%>
  <tr>
    <td><%=h cont.genre %></td>
    <td><%=h cont.level %></td>
    <td><%=link_to h(cont.name), :action => :show, :id => cont %></td>
  </tr>
<% end -%>
</table>
```

冒頭の<%= text_field_tag 'keyword', params[:keyword] %>
で文字入力フィールドがつくられ，そこに入れられた文字が params[:keyword]に格納され，

@conts = Cont.find(:all, :conditions => ['name like ?', '%' + params[:keyword] + '%'])
で，タイトルに検索ワードを含むcontが全て@contに格納される．その後，

```
<% @conts.each do |cont| -%>
  <tr>
    <td><%=h cont.genre %></td>
    <td><%=h cont.level %></td>
    <td><%=link_to h(cont.name), :action => :show, :id => cont %></td>
  </tr>
<% end -%>
```

で，@contがジャンル，難易度，タイトルの順に列挙され，タイトルにその項目の詳細ページへのリンクが張られる．

5.4　詳細

リスト：app/controllers/charts_controller.rb(一部)

```
  def show
    find_cont
  end

  def add_to_cart
    find_cont
    count = 1
    if count > 0
      cart = session[:cart]
      assoc = cart.assoc(@cont)
      if assoc then assoc[1] += count else cart.
        push [@cont, count] end
      flash[:notice] = "#{count} 個カートに入りました。"
    end
    redirect_to :action => 'show', :id => @cont
  end

  def view_file
    find_cont
    file = "pdf/#{@cont.id}"
    send_file(file, :filename => @cont.file_name, :stream => false, :disposition => 'in-
line')
```

```
  end

  before_filter :initialize_session_storage
  private
    def initialize_session_storage
      session[:cart] ||= []
    end

    def find_cont
      @cont = Cont.find(params[:id])
    end
```

リスト：app/views/charts/show.rhtml

```
<h2>詳細</h2>
<hr />

<%= form_tag :action => 'add_to_cart' %>
  <%= hidden_field_tag 'id', @cont.id %>
<dl>
  <dt>タイトル</dt>
  <dd><%= @cont.name %></dd>
  <dt>難易度</dt>
  <dd><%= @cont.level %></dd>
  <dt>ジャンル</dt>
  <dd><%= @cont.genre %></dd>
  <% if @cont.attribute_present?('file_name') %>
  <dt>File<dt>
  <dd>
    <%= link_to("[#{h(@cont.file_name)}]", :action => 'view_file', :id => @cont) %>
  </dd>
  <% end %>
</dl>
  <%= submit_tag 'カートに入れる' %>
<%= end_form_tag %>
<%= link_to 'カートを確認する', :action => 'cart' %><br>
<%= link_to '他にも選ぶ', :action => 'list' %><br>
<%= link_to '編集', :action => 'edit', :id => @cont %><br>
```

メソッド find_contは指定された項目をデータベースから見つけてくるメソッドで，多用されるためにメソッドとしてまとめてある． メソッドview_fileは添付されているファ

イルを見るという メソッドであり， メソッドadd_to_cartはカートにその項目を入れるメソッドである．本来は，入力フィールドを用意し，個数を入力できるようにしてあるが，本研究では必要ないと判断しcount = 1 と固定している．
また，privateの後にある def initialize_session_storageは，カートをセッション変数として格納している．これにより，選び続ける間，カートの内容を保持できている．

5.5　編集

リスト：app/controllers/charts_controller.rb(一部)

```
 def edit
   find_cont
 end

 def update
   set_file_name
   find_cont
   if @cont.update_attributes(params[:cont])
     save_file
     flash[:notice] = 'Cont was successfully updated.'
     redirect_to(:action => 'list')
   else
     render(:action => 'edit')
   end
 end

 def destroy
   find_cont
   @cont.destroy
   redirect_to(:action => 'list')
 end
```

リスト：app/views/charts/edit.rhtml

```
<h1>編集</h1>
<hr />

<%= start_form_tag({:action => 'update', :id => @cont },
                   :multipart => true) %>
 <%= render(:partial => 'form') %>
 <%= submit_tag "Edit" %>
<%= end_form_tag %>
<%= link_to 'Destroy', { :action => 'destroy', :id => @cont }, :confirm => 'Are you
```

```
sure?', :method => :post %><br/>
<%= link_to('back', :action => 'show', :id => @cont ) %>
```

ここは，新規登録と同じく_form.rhtmlを呼び入力フォームをつくり， find_contで指定されたContを読み込み，入力フォームにいれてある．Editのボタンをおすと_<%= start_form_tag({:action => 'update', :id => @cont }で，メソッドupdateが呼ばれて更新される．また，メソッドdestroyはデータベースからその項目を削除するメソッドであり， <%= link_to 'Destroy', { :action => 'destroy', :id => @cont }, :confirm => 'Are you sure?', :method => :post %>
で呼び出され，メッセージボックスが出て「Are you sure?」と訊いてから実行されるようになっている．

5.6 カート

リスト：app/controllers/charts_controller.rb(一部)

```
 def cart
   @cart = session[:cart]
   if @cart.empty?
     render :text => 'カートに何も入ってません。'
     return
   end
 end

  def remove_from_cart
   @cart = session[:cart]
   @cart.each do |assoc|
     item, count = assoc
     if item.id == params[:id].to_i
       @cart.delete(assoc)
       break
     end
   end
   redirect_to :action => 'cart'
 end

 def update_cart
   @cart = session[:cart]
   @cart.each do |assoc|
     cont, count = assoc
     count = params[cont.id.to_s].to_i
     assoc[1] = count
   end
```

```
    redirect_to :action => 'cart'
  end
```

リスト：app/views/charts/cart.rhtml

```
<h1>カートの確認</h1>

<%= form_tag :action => 'update_cart' %>

<table>
  <tr>
    <td>ジャンル</td>
    <td>難易度</td>
    <td>タイトル</td>
    <td>順序</td>
  </tr>
  <% i=1 %>
  <% @cart.each do |cont, count| %>
  <tr>
    <% count=i %>
    <% i += 1 %>
    <td><%=h cont.genre %></td>
    <td><%=h cont.level %></td>
    <td><%=h cont.name %></td>
    <td><%= text_field_tag cont.id, count, :size => 3, :maxlength => 3 %></td>
    <td><%= link_to '取消', :action => 'remove_from_cart', :id => cont %></td>
  </tr>
  <% end %>
</table>
  <%= submit_tag '順序を更新する' %><br/>
<%= end_form_tag %>

<%= form_tag :action => 'merge' %>
  <%= submit_tag '進む' %>
<%= end_form_tag %>
<%= link_to 'back', :action => 'list' %>
```

カートが空のときは，render :text => '〜〜'でメッセージを出力するようになっている．

カートは，商品をキーとし，個数を値とするハッシュとして表現できる．ただし，

Rubyはハッシュの順序を保持できないため，ここではカートを入れ子の配列として表現する．例えば，商品item1 を 5 個，商品item2 を 3 個入れたとき，カートは
cart = [[item1, 5], [item2, 3]]
となっている．

5.7　ファイルの統合

リスト：app/controllers/charts_controller.rb(一部)

```
  def merge
    d = 'pdftk '
    @cart = session[:cart]
    i = 0
    a = []
    @cart.each do |cont, count|
      a[count] = cont.id
      i += 1
    end
    j = 1
    while j <= i
      d += 'pdf/' + a[j].to_s + ' '
      j += 1
    end
    d += 'cat output merge.pdf'
    %x[#{d}]

    redirect_to :action => 'result'
  end

  def result
  end

  def look
    file = "merge.pdf"
    send_file(file, :filename => file, :stream => false, :disposition => 'inline' )
  end
```

リスト：app/views/charts/result.rhtml

```
<h1>クリックして保存してください</h1>
```

<%= link_to('merge.pdf', :action => 'look') %>

カートの確認からは，<%= form_tag :action => 'merge' %> でメソッドmergeが呼ばれ，順序に従ってpdfファイルを統合し，redirect_to :action => 'result'で自動的にresultにジャンプするようになっている．メソッドresultは何もせず，result.rhtmlを表示し，merge.pdf' をクリックすると，merge.pdfを表示するメソッドlookが呼ばれる．

第6章　考察

6.1　Ajax

Ajaxでは，従来のように，ブラウザからリクエストを受けたサーバーがレスポンスとして完全なWebページを返すのではなく，ページの再読み込みなしでサーバーと通信し必要な所だけ部分的に更新することができる．Ajaxには以下のような重要な点がある．

・Ajaxは必ずしもサーバーを呼び出す必要はなく，例えば，フォームに入力されたデータをチェックするという処理はクライアント側だけでできる．

・サーバーから返されるデータは，完全なページではなく，より小さな情報の断片である．したがって，データベースアクセスの回数は削減され，描画や通信にかかる時間も減少し，その結果，リクエスト全体にかかる合計時間も減少する．

・ユーザーインターフェースとサーバーから返されるレスポンスとの間に，直接の依存関係はない．通信はバックグラウンドで行われるため，その間もユーザーはページ上での作業を継続できる．

・ページとサーバーとの間で通信が発生しても，必ずしもユーザーインターフェースに対して変更を加える必要はない．

このように，Ajaxは従来のアプリケーションと比べて，応答性が大幅に向上している．

図5.1:Ajaxによる並べ替え

本研究では，並べ替えをDrag on Dropで行えるようにと，実装しようとしたが，実際に順序が変わらず．課題として残してしまった．

6.2　今後の課題

このWebアプリケーションには，まだ改良の余地がある．まず，前項で挙げたAjaxの実装である．Ajaxの特徴を考えると，並べ替えだけでなく検索ページ，ダウンロードページにも使うことができそうである．

次に，リストページである．上手い方法が見つからず，ビューもコントローラーも乱雑なコードになってしまった．

そして，まだまだ便利なパーツがあるかもしれないということである．Railsはまだまだ発展途上といえるので，今後，より便利なパーツが現れるかもしれないということはもちろん，現在あるパーツにも，さらに便利なものがある可能性もある．

さらに，最終発表で指摘していただいたことが２つある． １つは，目次である．ファイルの統合の際， １ページ目に目次を入れることができれば，多くの項目をダウンロードしても，目的の項目を素早く見つけることができる．

もう１つは，関連項目のあすすめである．このMapleチャート式には，それぞれの項目は決して独立しているものではなく，他の項目と関連している項目も多い．そういう項目をユーザーに対して提示できるなら，より使い易くなるだろう．

付録

付録1　pdftk

これは，ファイルの統合のときに使ったツールで．コマンドライン上でpdfを操作できる．

URL:http://www.accesspdf.com/index.php?topic=pdftk

上記のURLでダウンロードしてインストールした．

使い方

ファイルの連結：3 つのファイル in1.pdf、in2.pdf、in3pdf を連結して 1 つファイル out.pdf を生成する
pdftk in1.pdf in2.pdf in3.pdf cat output out.pdf
ワイルドカード（? とか *）を使うことも出来る
pdftk in?.pdf cat output out.pdf
pdftk *.pdf cat output out.pdf

ページ抽出：ファイル in.pdf の 3 ページから 5 ページまでと 9 ページ（従って、3, 4, 5, 9 の全 4 ページ）のみから成るファイル out.pdf を生成する
pdftk in.pdf cat 3-5 9 output out.pdf
または
pdftk A=in.pdf cat A3-5 A9 output out.pdf

ページ削除：ファイル in.pdf の 13 ページを取り除いたファイル out.pdf を生成する
pdftk in.pdf cat 1-12 14-end output out.pdf
または
pdftk A=in.pdf cat A1-12 A14-end output out.pdf

背景追加：ファイル in.pdf の全ページの背景に back.pdf を入れたファイル out.pdf を生成する
pdftk in.pdf background back.pdf output out.pdf

暗号化：素のファイル 1.pdf から、パスワード保護されたファイル 1.128.pdf【表示：OK、編集・印刷：要パス】を生成する（オーナー・パスワード を foo とする場合）
pdftk 1.pdf output 1.128.pdf owner_pw foo
更に、ユーザー・パスワード baz を設定して、表示・印刷を制限する
pdftk 1.pdf output 1.128.pdf owner_pw foo user_pw baz
ファイルを開くときにパスワード baz を要求するが、印刷は許可する
pdftk 1.pdf output 1.128.pdf owner_pw foo user_pw baz allow printing

復号化（解読）：パスワード保護されたファイル secured.pdf から保護機能を解除したファイル unsecured.pdf を生成する（オーナー・パスワード が foo の場合）
pdftk secured.pdf input_pw foo output unsecured.pdf

参照文献

[1] 吉田 和弘，馬場 道明 著，『ライド・オン・Rails』(ソフトバンク　クリエイティブ株式会社　2006)

[2] 高橋 征義 監修，かずひこ，喜多川 豪 著，『はじめようRuby on Rails』(株式会社アスキー　2006)

[3] arton 著，『かんたんRuby on RailsでWebアプリケーション開発』(株式会社翔泳社 2006)

[4] 伊尾木 将之，倉貫 義人，長瀬 嘉秀，松本 哲也 著，『実践 Ruby on Rails Webプログラミング入門』(株式会社ソーテック　2006)

[5] 西和 則 著，『Ruby on Rails入門　優しいRailsの育て方』(株式会社秀和システム 2006)

[6] Scott Raymond 著，牧野 聡 訳，『Ajax on Rails』(株式会社オライリー・ジャパン 2007)

[7] Brett McLaughlin 著，夏目 大，児島 修 訳，『Head Rush Ajax ー学びながら読むAjax入門』(株式会社オライリー・ジャパン　2006)

謝辞

本研究を遂行するにあたり，終始，多大なるご指導及び御教示を賜りました関西学院大学理工学部情報科学科教授西谷滋人先生に深く感謝の意を示すと共に，厚く御礼申し上げます.

最後に，それぞれ別の研究内容ではありましたが，関西学院大学理工学部情報科学科西谷研究室の皆様に心からの謝意を表します.