[/home/s1211436/www_local/wp/enquete_form2.rb]

```
#!/usr/bin/env ruby
# encoding: utf-8

begin

require 'cgi'
require 'sqlite3'
require 'logger'

@cgi = CGI.new
@questions = Array.new
@log = Logger.new("/home/s1211436/wp_log.txt")
@log.progname = __FILE__
@log.level = Logger::DEBUG

def read_questions_sql (database)
    SQLite3::Database.new(database) do |db|
        db.transaction() do
            lines = db.execute("select * from questions;")
            lines.each do |line|
                line = CGI.escapeHTML(line[0])
                next if line.empty?
                @questions << line
            end
        end
    end
end
```

一連の処理を関数にまとめるとコードの見通しが良くなると授業の最初に話があったので実践した。

←@log.progname = __FILE__ とすると楽だ。

←アンケートの選択肢もSQLiteに格納した。
SQLは以下の通りである。

```
create table questions (
        question text
);
.import question.txt questions
create table votes (
        name text
);
```

選択肢を書いたquestion.txtの中身は、例えば以下のように書いてある。

```
どれをやりたいですか(複数回答可)
COJT ハードウェア
COJT ソフトウェア
普通の情報メディア実験
退学
転学類
その他
```

```
def print_questions()
    str = String.new
    while question = @questions.shift
        str = str + <<-EOF
            <div><label>
                <input type='checkbox' name='jobs'
value='#{question}'>#{question}
            </label></div>
        EOF
    end
    return str
end
```

←HTML出力がバラバラになると見づらいので、HTMLを出力するにはprint_html()を呼び出せば良いようにした。
アンケートの選択肢をループで出力する部分は無理にprint_html()に埋め込むと見づらいので、print_questions()に分けて式展開した。

はじめは、print_questions()で文字列をreturnするのではなく、文字列をprint()していたため、どうしてHTMLの先頭にアンケートの選択肢が飛び出してしまうのか悩んだ。式展開が、標準出力ではなく、返り値を展開することに気がついた。

```
def print_html()
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <!doctype html>
        <html lang="ja">
        <head>
            <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
            <title>投票システム</title>
        </head>
        <body>
            <h1>投票システム</h1>
            <form action="vote2.rb" method="post">
                <h2>#{@questions.shift}</h2>
                #{print_questions()}
                <div><input type="submit" value="送信"><input type="reset" value="クリア"></div>
            </form>
            <p><a href="view_result2.rb">投票結果を見る</a></p>
        </body>
        </html>
    EOF
end

read_questions_sql("report1025.db")
print_html

rescue => ex
    @log.debug(ex)
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <html><body><pre>
        #{ex.message}
        #{CGI.escapeHTML(ex.backtrace.join("\n"))}
        </pre></body></html>
    EOF
end
```

[/home/s1211436/www_local/wp/vote2.rb]

```
#!/usr/bin/env ruby
# encoding: utf-8
begin

require 'cgi'
require 'logger'
require 'sqlite3'

@cgi = CGI.new
@log = Logger.new("/home/s1211436/wp_log.txt")
@log.progname = __FILE__
@log.level = Logger::DEBUG
@questions = Array.new

def read_questions_sql (database)
    SQLite3::Database.new(database) do |db|
        db.transaction() do
            lines = db.execute("select * from questions;")
            lines.each do |line|
                line = CGI.escapeHTML(line[0])
                next if line.empty?
                @questions << line
            end
        end
    end
end

def parameters_check_sql
    @jobs = @cgi.params["jobs"]
    @jobs.each do |job|
        if @questions.include?(job)
        else raise "存在しない選択肢が送信されました。"
        end
    end
end

def write_db (database)
    SQLite3::Database.new(database) do |db|
        db.transaction() do
            @jobs.each do |job|
                next if job.empty?
                db.execute("insert into votes values(?);", job)
            end
        end
    end
end
```

← 前回のvote.rb は、
1. テキストファイルをロックしてする。
2. パラメータチェックを行う。
3. 書き込み処理をする。
4. ロックを解除する。
の順番で処理していた。

しかし、不正な選択肢が1つでもあった場合に書き込み処理が行われないように、パラメータチェックが全て終わってから書き込み処理をしていたので、
1. パラメータチェックを行う
2. データベースのトランザクションを開始する。
3. 書き込み処理をする。
4. トランザクションを終了する。

の順番の方が、ロックされている時間が少なくて済むだろう。関数も2つに分けた。

```
def print_html (title)
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <!doctype html>
        <html lang="ja">
        <head>
            <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
            <title>#{title}</title>
        </head>
        <body>
        <p>#{title}</p>
        <p><a href="view_result2.rb">投票結果を見る</a></p>
        <p><a href="enquete_form2.rb">投票に戻る</a></p>
        </body>
        </html>
    EOF
end

read_questions_sql("report1025.db")
@questions.shift
parameters_check_sql
write_db("report1025.db")
print_html("投票ありがとうございます。")

rescue => ex
    @log.debug(ex)
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <html><body><pre>
        #{ex.message}
        #{CGI.escapeHTML(ex.backtrace.join("\n"))}
        </pre></body></html>
    EOF
end
```

[/home/s1211436/www_local/wp/view_result2.rb]

```
#!/usr/bin/env ruby
# encoding: utf-8

begin

require 'cgi'
require 'logger'
require 'sqlite3'

@cgi = CGI.new
@log = Logger.new("/home/s1211436/wp_log.txt")
@log.progname = __FILE__
@log.level = Logger::DEBUG
@questions = Array.new
@h = Hash.new(0)

def read_questions_sql (database)
    SQLite3::Database.new(database) do |db|
        db.transaction() do
            lines = db.execute("select * from questions;")
            lines.each do |line|
                line = CGI.escapeHTML(line[0])
                next if line.empty?
                @questions << line
            end
        end
    end
end
```

← 集計はとりあえずRubyで処理した。

```
def count (database)
    SQLite3::Database.new(database) do |db|
        db.transaction() do
            lines = db.execute("select * from votes;")
            lines.each do |line|
                line = CGI.escapeHTML(line[0])
                next if line.empty?
                @h[line] += 1 if @questions.include?(line)
            end
        end
    end
end
```

← 集計結果も、enquete_form2.rb と同様に、HTML出力が1つにまとまるようにした。ループ部分は独立させて、HTMLの文字列を返すようにしている。

```
def print_count ()
    count = 0
    str = "<ul>\n"
    @h.each do |key, value|
        str = str + "<li>#{key}: #{value}</li>\n"
        count += value
    end
    str = str + "</ul>\n"
    str = str + "<p>得票数 = #{count}</p>\n"
    return str
end
```

```
def print_html (title)
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <!doctype html>
        <html lang="ja">
        <head>
            <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
            <title>#{title}</title>
        </head>
        <body>
        <h1>#{title}</h1>
        <h2>#{@theme}</h2>
        #{print_count()}
        <p><a href="enquete_form2.rb">投票に戻る</a></p>
        </body>
        </html>
    EOF
end

read_questions_sql("report1025.db")
@theme = @questions.shift
count("report1025.db")
print_html("投票結果")

rescue => ex
    @log.debug(ex)
    print @cgi.header("text/html; charset=utf-8")
    print <<-EOF
        <html><body><pre>
        #{ex.message}
        #{CGI.escapeHTML(ex.backtrace.join("\n"))}
        </pre></body></html>
    EOF
end
```

授業の感想

1.
配列とハッシュの補足(スライド4枚目)の処理がよく分からなかった。特にsortのブロック変数がわからない。
eachのブロック変数もわからない。

```
h = {"z" => 12, "b" => 34, "a" => 56, "c" => 78}    # "c"を追加した。
h.to_a.sort {|m,n|
    p (m[0] <=> n[0])
}.each {|e|
    puts "#{e[0]} = #{e[1]}"
}
```

これを実行すると、sortブロックのブロック変数には次の様な順番で代入される。

```
m = ["z", 12], n = ["a", 56]
m = ["a", 56], n = ["c", 78]
m = ["z", 12], n = ["c", 78]
m = ["b", 34], n = ["c", 78]
m = ["a", 56], n = ["b", 34]
```

また、<=>演算子は左が小さければ-1を、大きければ1を返すようだが、これが次のeachブロックのブロック変数に、なぜ ["a", 56], ["b", 34], ...と配列を渡していくのか分からない。

2.
スペースを入れても良いヒアドキュメントを知らなかった。気持ち悪いインデントが解消されて嬉しい。
Rubyをちゃんと自習しないと不便なようだ。