

CS306: Introduction to Perl

Web and Database
Applications

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

The CGI Model

1. The web server gets a request for a Perl CGI script.
2. The script is executed, meaning that the Perl interpreter is launched, and the web server hands over the request and data based on the CGI protocol.
3. STDOUT is redirected back through the CGI gateway. The script produces its output and exits. The Perl interpreter terminates.

Dynamic Web Apps With CGI.pm

- CGI is the Common Gateway Interface
- The simplest way to provide interactive applications on the web
- Stateless - works as http requests. Request received from browser, response sent to browser, connection closed, program terminated.
- Perl provides CGI.pm to create CGI scripts

CGI.pm - Simple Example

- ```
use CGI qw/:standard/;

print header, start_html('Custom Doc Quoter'),
 h1('Doc Quoter'),
 start_form,
 "Enter a number: ", textfield('num'),
 p,
 submit, end_form, hr;

if (param()) {
 print param(num) . " gigawatts!!! ",
 "The only place to get " . param(num) .
 " gigawatts is a bolt of lightning!!";
}
```

## CGI.pm Features

- CGI.pm provides a lot of helper functions that produce HTML as output. Many take arguments... header(), h1('heading'), p, start\_form, start\_html('title'), etc.... many correspond to the equivalent HTML tag.
- CGI.pm gathers up all the CGI input data (passed by either the GET or POST method) and exposes it through the param() function for you to get at it.

## Sticky Forms

- Sticky forms mean that the form fields are pre-populated with their previous values if you revisit them.
- The CGI.pm html shortcuts like textfield() provide this for free. Can be especially useful if you are going back to a form to ask a user to correct a mistake - they don't have to type everything again.

## A General CGI Framework

- ```
use CGI qw/:standard/;

# Do stuff for ALL HTML pages here...
# Setup header, page title, etc...

if (param()) {
    # User just submitted the form
    # process the data and put up a response page
} else {
    # User is surfing to the page for the
    # first time
    # show them a fill-in form/options menu/etc...
}
```

Passing Data Between Pages

- Since HTTP is stateless and the Perl interpreter ends, you can't keep variables in memory across multiple page submissions.
- Two strategies:
 - Embed the data back into the page in hidden form fields. Simpler, if a bit clunky.
 - Store the data server-side in some persistent form. Harder, because you have to differentiate between multiple clients with a session key or similar.

Client-Side Data Persistence

- Generate hidden form fields with CGI.pm's `hidden()` function. Read the perldoc.
- Safe and easy because each client then hands the data back in on the next request, so you know exactly who it is coming from.

Server-Side CGI Data Persistence

- One way to do server-side persistence is to use CGI.pm's ability to set a cookie in the browser.
- Generate a unique ID, store it in a cookie, and then check for that cookie every time the script is hit.
- If cookie found, can then retrieve data previously stored on disk tagged with the unique key
- Requires user to accept cookies

Some Notes on CGI

- The CGI script is called with a URL like `http://www.cis.uab.edu/fran/sample.cgi`
- The web server has to allow the `fran/` web directory to run CGI scripts
- The web server must be configured to recognize the `.cgi` extension as a CGI script
- `sample.cgi` must be set to be executable by the user running the web server (mode 755)

More CGI.pm Usage Notes

- Make sure you speak HTML. Browsers don't understand `\n`, they understand `
`
- Always print out a proper header (and only one). It's generally appropriate to do this near the top of your script, not inside each case.

Debugging CGI.pm Scripts

- If you run them from the command-line, they allow you to pass name=value pairs to represent the incoming data.
- Hit Ctrl-D when you are done setting input data, and the script will run. You'll see what it would have sent to the browser.

Learn More About CGI

- perldoc CGI for the CGI.pm documentation
- Lincoln Stein's CGI.pm book - only of the only books published specifically for one Perl module.
- The Perl Cookbook has a chapter on CGI programming

More Debugging

- use CGI::Carp qw(fatalsToBrowser);
- This will redirect errors to the browser for easier debugging if you do not have access to the web server's error log.
- Read the CGI::Carp documentation for ways to redirect the errors to a file instead.

Simple Databases

- Three levels of databases...
 - Plain text files in some user-defined format
 - File-based databases like BerkeleyDB (DB)
 - SQL engines like MySQL, Oracle, MS-SQL, etc...
- You already know how to implement the first
- We will talk about the 2nd and 3rd methods

DBM

- DBM actually refers to a variety of file-based database formats (NDBM, DB, GDBM, SDBM, ODBM)
- Different platforms had different DBM implementations so Perl needed multiple interfaces to support all of them.
- Perl ships with SDBM so that's guaranteed to be everywhere, so there is always some form of DBM available.

DBM Example

- ```
use AnyDBM_File;
dbmopen(my %stocks, "ticker", 0666);
print "Enter a stock ticker symbol: ";
chomp(my $sym = <STDIN>);
print "Enter a price for $sym: ";
chomp(my $price = <STDIN>);
$stocks{$sym} = $price;
for (keys %stocks) {
 print "$_ is $stocks{$_}.\n";
}
dbmclose(%stocks);
```
- Each run through this will add another stock symbol and price to the DBM

## DBM Abstraction

- All of these different implementations made it difficult to write portable programs
- Perl now has AnyDBM\_File, which will use whatever is available locally.
- The general philosophy of DBM files is that you open them, work on them just like a hash, and then save them to disk when done. They can then be retrieved during a later run.

## DBM Limitations

- DBMs are designed to store simple key=value pairs. Most can't store references to data more complex than scalars (they'll actually store the string "HASH{0x2bbc4a}" instead)
- Solution: make multiple DBMs, or use MLDBM. It uses Data::Dumper to generate storable strings from complex data. It's not standard, though.
- DBMs also have record size limits (1k is safe)

## Emptying a DBM File

- Quite easy....
- ```
dbmopen (my %stocks, 'ticker', 0666);  
%stocks = ();  
dbmclose(%stocks);
```

SQL Databases - DBI

- Perl provides an extremely robust abstraction layer called the DBI which allows you to write database-independent code to execute SQL queries.
- With the DBI, it is very easy to port applications between databases, and the syntax remains the same no matter which database you are using.

Sorting Large DBM Files

- It's worth noting that DB_File (BerkeleyDB) has a BTREE mode which will store your data as a binary tree. This is much more efficient for large data sets.
- Now, calls to keys(), values() and each() come back ordered automatically.
- You can provide a comparison function to tell DB_File how to sort the data.

DBI Example

- ```
use DBI;
my $dbh = DBI->connect('DBI:driver:database',
 'username',
 'auth',
 {RaiseError => 1,
 {AutoCommit => 1, }});

$dbh->do($sql);
my $sth = $dbh->prepare($sql);
$sth->execute();
while (my @row = $sth->fetchrow_array) {
 # do something
}
$sth->finish();
$dbh->disconnect();
```

## DBD

- The DBI works by sitting on top of a database-specific DBD driver library. There are DBD modules available for just about all popular databases, including MySQL, PostgreSQL, Oracle, and ODBC (for MS and other databases).
- Specify in the connect string:  
my \$dbh = DBI->connect('DBI:mysql:dbname'...