HyperText Transport Protocol
HTTP
Charles Severance
csev@umich.edu

Textbook: Using Google App Engine, Charles Severance

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.
http://creativecommons.org/licenses/by/3.0/.
Copyright 2009, 2010, Charles Severance, Jim Eng

Making an HTTP request

• Connect to the server
  • a "hand shake"
• Request a document (or the default document)
  • GET http://dr-chuck.com/pageI.htm
  • GET http://www.mlive.com/ann-arbor/
  • GET http://www.facebook.com
http://www.dr-chuck.com/page1.htm

protocol  host  document

The First Page
If you like, you can switch to the Second Page.
An HTTP request - response cycle
Making an HTTP request

- Connect to the server
  - a "hand shake"
- Request a document (or the default document)
  - GET http://dr-chuck.com/page1.htm
  - GET http://www.mlive.com/ann-arbor/
  - GET http://www.facebook.com

Getting Data From The Server

- Each time the user clicks on an anchor tag with an href= value to switch to a new page, the browser makes a connection to the web server and issues a "GET" request - to GET the content of the page at the specified URL.
- The server returns the HTML document to the Browser which formats and displays the document to the user.
5. Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use. For backwards compatibility with the more limited HTTP/0.9 protocol, there are two valid formats for an HTTP request:

\[
\text{Request} = \text{Simple-Request} | \text{Full-Request} \\
\text{Simple-Request} = "GET" \text{ SP Request-URI CRLF} \\
\text{Full-Request} = \text{Request-Line} ; \text{Section 5.1} \\
\quad \text{| General-Header} ; \text{Section 5.2} \\
\quad \text{| Request-Header} ; \text{Section 5.3} \\
\quad \text{| Entity-Header} ) ; \text{Section 7.1} \\
\quad \text{| CRLF} \text{ } [ \text{ Entity-Body } ] ; \text{Section 7.2}
\]

If an HTTP/1.0 server receives a Simple-Request, it must respond with an HTTP/0.9 Simple-Response. An HTTP/1.0 client capable of receiving a Full-Response should never generate a Simple-Request.

5.1 Request-Line

The Request-Line begins with a method token, followed by the

\[
\text{Request-URI} \text{ CRLF}
\]

$ telnet www.facebook.com 80
Trying 69.63.187.19...
Escape character is '^]'.
GET /
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
  id="facebook" class="no_js">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-language" content="en" />
    <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
    <script type="text/javascript">

Hmmm - This looks kind of Complex.. Lots of GET commands

Firebug helps again

- If you haven’t already installed Firebug, you need it now
- It can help explore the HTTP request-response cycle
- Some simple-looking pages involve lots of requests:
  - HTML page(s)
  - Image files
  - CSS Style Sheets
  - Javascript files
Sending Data to an Application

Forms - Input on the Web

Why do we call them forms?
Forms Need Servers

- Forms effectively gather data from the user and "submit" it to a web page on a server
- The earliest form of server-side processing was called GGI - Which stood for Common Gateway Interface
- CGI allows software to "receive" the input parameters and produce the HTML response - rather than simply reading the HTML content from a file


Using Forms Without Servers

- Submitting form data works without a server - the browser moves from static page to static page
- But the data from the forms is neither saved no is it usable
So let's write some code and put it in a server...

```python
print "Your guess is", guess
answer = 42
if guess < answer :
    print "Your guess is too low"
if guess == answer :
    print "Congratulations!"
if guess > answer :
    print "Your guess is too high"

<p>Your guess is 20.
Your guess is too low.
</p>
<form method="post" action="/">
    Enter Guess:
    <input type="text" name="guess" />
</form>
```
Your guess is 20.
Your guess is too low.

<form method="post" action="/">
  Enter Guess:
  <input type="text" name="guess" />
</form>

Attributes of a form element

- "action" attribute tells where to submit the form
  - Usually the path to a script or program on the server that processes the form inputs

- "method" attribute tells how to submit the form
  - In this case using HTTP POST

See page 30 of RFC 1945
Your guess is 20.
Your guess is too low.
<form method="post" action="/"
  
  Enter Guess:
  <input type="text" name="guess" />

  <input type="submit" />
</form>

GET .vs. POST

- Two ways the browser can send parameters to the web server
- GET - Parameters are placed on the URL which is retrieved
- POST - The URL is retrieved and parameters are appended to the request in the the HTTP connection

Intended purpose of POST

- Posting a message to a bulletin board, newsgroup, mailing list
- Annotation of existing resources
- Extending a database through an append operation
- Creating a new object
- Providing a block of data, such as the result of submitting a form
As opposed to GET

- Retrieve a resource identified by the path portion of the URL

Normal use of GET

Browser

Web Server

GET http://www.dr-chuck.com/page2.htm

<h1>The Second Page</h1>
<p>If you like, you can switch back to the <a href="page1.htm">First Page</a>.</p>

Browser

Web Server

Browser
Passing Parameters with GET

GET /simpleform.html?guess=25
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14

POST /simpleform.html
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 13
guess=25

"Rules" for GET and POST

- GET is used when you are reading or searching things
- POST is used when data is being created or modified
- Web search spiders will follow GET URLs but generally not POST URLs
- GET URLs should be "idempotent" - the same URL should give the "same thing" each time you access it
- GET has an upper limit of the number of bytes of parameters and values (think about 2K)
Advanced Form Fields

Fieldset and Legend

```html
<form method="get" action="/welcome">
<fieldset>
<legend>All About You</legend>
<p>
Enter your name: <input type="text" name="yourname" />
</p>
<p><input type="submit" /></p>
</fieldset>
</form>
```

Input Types

- Text
- Password
- Checkbox
- Radio Button
- Hidden
- Submit
- File

Text Input

```html
<p>
Enter your name: <input type="text" name="yourname" />
</p>
<p>
Enter your nickname: <input type="text" name="nickname" value="Bob" />
</p>
```

The `name=attribute` is the parameter name used to submit the data to the server.

Text fields can either start out blank or have content pre-populated.
Password Input Type

Your password: 
<input type="password" name="password" />

This only hides the password from view on the screen - to protect the password while in-transit, you need to send the data over https.

Hidden

- Hidden fields are used generally so that a program in a web server can send some internal information back to itself.

<input type="hidden" name="peekaboo" value="hereiam" />

Radio Buttons - Choice

- In the morning
- In the afternoon
- In the evening

<p>
<input type="radio" name="timeslot" value="morning" checked="on" />
In the morning<br />
<input type="radio" name="timeslot" value="afternoon" />
In the afternoon<br />
<input type="radio" name="timeslot" value="evening" />
In the evening
</p>

Sent to server  timeslot=morning

Checkbox - Multiple Select

- I have read the terms and conditions.
- I agree that you can contact me regarding special offers in the future.

<input type="checkbox" name="terms" value="yes" checked="on" />  terms=yes
<input type="checkbox" name="offers" value="yes" />  offers=yes
Drop Down List

Which best describes you?  

```html
<p>
Which best describes you?
<select name="role">
<option value="1">Secretary</option>
<option value="2" selected="selected">Web Designer</option>
<option value="3">Manager</option>
<option value="4">Cleaner</option>
<option value="5">Other</option>
</select>
</p>
```

A drop-down list generates a single value when it is sent to the server.

```
role=2
```

Textarea for paragraphs

```html
<p>Please tell us about your hobbies:
<textarea name="hobbies" rows="7" cols="40">
Old Value
</textarea>
</p>

Textareas can become rich text areas - http://tinymce.moxiecode.com/

Submit Button(s)

```html
<p><input type="submit"/></p>

<!-- Multiple submit buttons -->
<p>
<input type="submit" name="subtype" value="submit"/>
<input type="submit" name="subtype" value="cancel"/>
</p>
```

When you have multiple submit buttons the value can be used to figure out which button was pressed.

Writing Your AppEngine Application
print "Your guess is", guess
answer = 42
if guess < answer :
    print "Your guess is too low"
if guess == answer :
    print "Congratulations!"
if guess > answer :
    print "Your guess is too high"

- app.yaml - Defines the name of your application and the high level routing of incoming URLs
- index.py - The code for your application
- index.yaml - Created by App Engine

app.yaml
- The app.yaml file routes requests amongst different Python scripts.
  - application: ae-01-guess
    version: 1
    runtime: python
    api_version: 1
  handlers:
  - url: /.*
    script: index.py
import sys

print 'Content-Type: text/html'
print "
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
    guess = int(data[data.find('=')+1:])
except:
    guess = -1
print 'Your guess is too high'

import sys

print 'Content-Type: text/html'
print "
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()
# print data
try:
    guess = int(data[data.find('=')+1:])
except:
    guess = -1
print 'Your guess is too high'
import sys
print 'Content-Type: text/html'
print ''
print '<pre>'

# Read the form input which is a single line as follows
# guess=42
data = sys.stdin.read()

# print data
try:
    guess = int(data[data.find('=')+1:])
except:
    guess = -1

print guess
print ''
print '</pre>'
Assignment

- http://code.google.com/appengine/docs/python/gettingstarted/
- Introduction
- Development Environment
- Hello World! - Change it to "Hello Yourname"

Summary

- We can present our user a form with fields to fill in.
- The data from the form can be sent to the server of our choice
- We write an application on the server which received the data and produces a response