

HTTP 6

Java Regular Expression Addendum

A few more cases I did not get to last time...

Passing the constant `Pattern.MULTILINE` to `Pattern.compile()` enables "multiline" mode.

In that case `^` matches the start of each line and `$` matches a point just before the eol for each line.

The constant `Pattern.CASE_INSENSITIVE` makes the pattern matching not case sensitive. Pass multiple constants by combining them with `+` or bit-wise or `|`

Use `matcher.replaceAll(str)` to replace all occurrences of a pattern with a string. As we saw before, to get a `\s` in the pattern, use `\\s` in the Java string, since the `\` needs to be escaped.

To get a `\"` in the pattern, need `\\\"` in the Java string. This is required since the `\` needs to be escaped, but so does the `"`, since it normally marks the end of the string.

Persistent Connections -- HTTP 1.0

In HTTP 1.0, connections close by default.

With HTTP 1.0 we can get away without sending `content-length`, since the EOF will mark the end of the content.

The `connection:keep-alive` header can be used with HTTP 1.0, but it's a kludge. HTTP 1.1 has real persistent support.

Persistent Connections -- HTTP 1.1

With HTTP 1.1, persistent is the default

Therefore, we must use `content-length`: in the response

The `connection:close` may be used in either request or response to indicate the intention to close the connection at the end of the request/response

Not Pipelined Client Code

A way of thinking about pipelining is to think about the client code

Not pipelined -- client waits for each response before sending next request.

Lower bound of $n * \text{latency} * 2$ for n requests

// suppose we have a `paths` array of request

```
for (int i=0; i<paths.length; i++) {  
    request(paths[i]);  
    processResponse();  
}
```

Pipelined Client Code

The pipelined version simply does not wait for response-n to show up before sending request-n+1.

Much more efficient with respect to latency

Lower bound of latency*2 + the bandwidth costs

```
// send all the requests
for (int i=0; i<paths.length; i++) {
    request(paths[i]);
}

// now turn around and process all the responses
for (int i=0; i<paths.length; i++) {
    processResponse();
}
```

HEAD Request

Request sent to server, instead of GET

HEAD / HTTP/1.0

Like GET, but just gets the header, not the body/content.

Handy to see if a resource is there or what type it is, but without actually downloading it

Fancy Server Interpretation of Path

The server can interpret the request path however it would like.

We saw the ~user example, but an HTTP server can be even more free-form.

For example, we could write a quote server that took request paths like the following...

/fish/wilde

The server is not required to map that into the file system.

The server could separate out the words ("fish", "wilde"), search through a local database of quotes, and then send back a minimal HTTP response of the quotes in HTML.

This requires no special behavior by the browser -- it has a URL, makes a request, gets back some HTML to display.

The urls look normal -- `http://www.foo.com/fish/wilde` -- can be bookmarked, emailed to friends, etc. etc.

```
HTTP/1.0 200 OK
```

```
Content-type:text/html
```

```
<html>
```

```
<ul>
```

```
<li>Even a fish could stay out of trouble if it would learn to keep its mouth shut.
```

```
<li>Biography lends to death a new terror. (Oscar Wilde)
```

```
...
```

Caching

Copy of recent requests kept nearer to client. First try to satisfy request out of cache.

Browser keeps local cache in memory + on local disk.

Proxy cache -- one or more machines between the client and the server. The browser is set to make its requests to the proxy (see "proxy" in your browser prefs). The proxy will re-issue the request to the real "origin" server if needed. "squid" is a popular open source proxy cache (<http://www.squid-cache.org/>)

Browser Cache

Browser cache -- keep copies of HTML, GIF, ... seen recently

Back button...

Could just show the cached content, or could do an if-modified-since request...

Cache Control

May be used in either request or response

Cache-Control: no-cache ## forbid caching

Cache-Control: max-age=600 ## limit caching to the given number of secs

Expires: date ## a time, or "0" = no cache, this is the old way

Cache Failure

Because of earlier bad cache implementations, many sites needlessly specify no-caching.

They got tired of tech-support calls where people were seeing "old" content.

This unfortunate situation makes the (common) back-button case slower than need be if the content creators and web browsers were all following the cache spec as originally defined.

There's an interesting analogy to the prisoner's dilemma -- both the clients and servers are "defecting" from the win-win standard.

Cache Effects

Advantages

Helps with back button (browser cache). Ideally, hitting the back button, could just show the state as it appeared earlier very quickly -- get everything from local cache, and make no network traffic. The reality is more complex, since some content is marked as "no-cache".

Helps common web items, e.g. little graphic buttons on yahoo main page, cached for all users

Gives better performance -- a cache on your local LAN is much faster than the Internet at large

Reduces the "upstream" bandwidth bill of an ISP

Disadvantages

More complex -- has a reputation of creating tech support hassles, so some servers turn it off for all pages.

Could screw up unique pages, such as a view of your shopping cart, that only make sense for one user, and which change over time. The system needs to know to not cache a page like that.

Screws up "hit counting" on the origin server. (solvable)

Cache vs. CGI

Browsers may use different caching/back-button strategies depending on the type of url.

Displaying plain pages from cache, but re-doing the requests when going back to a url like /cgi-bin/cart.pl?add=foo

We will have to worry about this case when doing CGIs

Cache Technology

HTTP includes many features, directives, etc. to facilitate caching. Clients and servers can indicate: the last mod time of an element, whether something should be cached and for how long

The most common optimization is the "If-modified-since" GET that checks if the cached version is up to date. Saves bandwidth with images. (One reason web designers err towards image-heavy pages. The images load up the designers cache, so they don't see how terribly slow the page is on the first-hit for users at large.)

Cache server can communicate "number of hits" back to the origin server with the "meter" header

HTTP Proxy

A machine between the client and the server

Cache is most common application

1. Could be used to filter content
2. Could be used to contact the origin sever in a special way -- say over an encrypted channel. Or say for example, if www.maosucks.com is blocked from inside China. The proxy contacts the blocked site and relays the content back to the client.
3. Could re-write the content in some way

1. Browser Proxy

Browser has a formal "proxy" preference -- put in the proxy machine:port, and the browser will make all its requests to the proxy, instead of the origin server of the content.

The browser will make a request like the following to the proxy. Note that the full URL is given as the request path, not just the path. This is peculiar to proxying.

```
GET http://www.yahoo.com/ HTTP/1.1
Host: www.yahoo.com
...
```

The proxy contacts, in this case, www.yahoo.com, and then relays the response back to the browser. Along the way, the proxy can do caching, editing of the content, ...

2. URL Proxy Filtering

The URL points to the proxy machine, and the desired URL is passed as an argument on that URL, typically as a query string

`http://www.degraeve.com/scripts/babel2/babel.cgi?d=piglatin&url=http://www.yahoo.com`

The browser makes its request in the usual way to the proxy server, it turns around makes the real request, and then forwards the results back to the client

Pig-Latin Example

What does our course page look like translated into pig-latin?

`http://www.degraeve.com/scripts/babel2/babel.cgi?d=piglatin&url=http://www.stanford.edu/class/cs193i/`

CS193i Internetyay Echnologiestay

Elcomeway otay CS193i orfay Ingspray 2002.

....

It translates the HTML, and also updates all the href= URLs to also go through the proxy. Here is our handouts link...

```
<a
href="http://www.degraeve.com/scripts/babel2/babel.cgi?d=piglatin&url=http://www.sta
nford.edu/class/handouts023/">Andoutshay</a>
```

There are many such URL proxy filter "services" on the net -- Swedish chef, ... This is a form of server-side CGI program, which is our next topic.

http-equiv

In the HEAD section of the HTML

A low-budget way of communicating HTTP like things to the browser

```
<html>
```

```
<head>
```

```
  <meta http-equiv="cache-control" content="no-cache">
```

```
</head>
```

Refresh

Read by the client

```
<META HTTP-EQUIV="Refresh" CONTENT="seconds; URL=url">
```

Can re-load the page on some schedule (URL is optional)

Not an official part of HTML -- a cheesy extension

Or can "bounce" to another page -- a quick-n-dirty technique if you need to

bounce the client to another page. Not as clean as a real 301 re-direct, but much easier, since it only requires HTML.

This technique can mess up the function of the back button -- bad style

Web Log Example

Major fields: IP addr, [date], request, result code, length, referer

Notice: see the IP addr and time of the request

Notice: request for HTML, followed by req for GIF in that page

Notice: IE bug where it requests the PDF three times in a row

Notice: the google referer

Notice: the favicon.ico -- the little icon for bookmarking

```

128.205.181.102 - - [26/Apr/2002:11:58:55 -0700] "GET / HTTP/1.1" 200 6964
"http://www.cse.buffalo.edu/~alphonce/Courses/Spring2002/cse116/resources.html"
128.205.181.102 - - [26/Apr/2002:11:58:56 -0700] "GET /smallbinky.gif HTTP/1.1" 200
7648 "http://cslibrary.stanford.edu/"
128.205.181.102 - - [26/Apr/2002:11:59:09 -0700] "GET /112/ HTTP/1.1" 200 2297
"http://cslibrary.stanford.edu/"
128.205.181.102 - - [26/Apr/2002:11:59:09 -0700] "GET /112/screen.jpg HTTP/1.1" 200
20070 "http://cslibrary.stanford.edu/112/"
141.153.141.5 - - [26/Apr/2002:11:59:36 -0700] "GET /103/ HTTP/1.1" 200 2544
"http://cslibrary.stanford.edu/"
141.153.141.5 - - [26/Apr/2002:11:59:48 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 200 32768 "http://cslibrary.stanford.edu/103/"
141.153.141.5 - - [26/Apr/2002:11:59:49 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 206 45556 "-"
141.153.141.5 - - [26/Apr/2002:11:59:51 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 206 17888 "-"
128.12.106.149 - - [26/Apr/2002:11:59:53 -0700] "GET /108/EssentialPerl.pdf
HTTP/1.1" 304 - "http://smi.stanford.edu/projects/helix/bmi214/"
128.205.181.102 - - [26/Apr/2002:11:59:59 -0700] "GET /112/JTetris.jar HTTP/1.1" 200
31498 "http://cslibrary.stanford.edu/112/"
128.205.181.102 - - [26/Apr/2002:12:00:52 -0700] "GET /112/Readme.txt HTTP/1.1" 200
5090 "http://cslibrary.stanford.edu/112/"
143.166.99.177 - - [26/Apr/2002:12:01:01 -0700] "GET /105/ HTTP/1.0" 200 3212
"http://www.google.com/search?hl=en&q=linked+list"
209.49.118.20 - - [26/Apr/2002:12:01:02 -0700] "GET /105/ HTTP/1.0" 200 3212
"http://learnfree.stanford.edu/103/"
143.166.99.177 - - [26/Apr/2002:12:01:11 -0700] "GET /105/LinkedListProblems.pdf
HTTP/1.0" 200 32768 "http://cslibrary.stanford.edu/105/"
128.205.181.102 - - [26/Apr/2002:12:01:23 -0700] "GET /112/ HTTP/1.1" 304 -
"http://cslibrary.stanford.edu/"
128.205.181.102 - - [26/Apr/2002:12:01:23 -0700] "GET /112/screen.jpg HTTP/1.1" 304
- "http://cslibrary.stanford.edu/112/"
141.153.141.5 - - [26/Apr/2002:12:02:19 -0700] "GET /favicon.ico HTTP/1.1" 404 296
"-_"
129.110.47.188 - - [26/Apr/2002:12:04:23 -0700] "GET / HTTP/1.1" 200 6964
"http://www.google.com/search?hl=en&q=linked+list+uses"
129.110.47.188 - - [26/Apr/2002:12:04:24 -0700] "GET /smallbinky.gif HTTP/1.1" 200
7648 "http://cslibrary.stanford.edu/"
129.110.47.188 - - [26/Apr/2002:12:04:37 -0700] "GET /103/ HTTP/1.1" 200 2544
"http://cslibrary.stanford.edu/"
129.110.47.188 - - [26/Apr/2002:12:04:41 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 200 46681 "http://cslibrary.stanford.edu/103/"
129.110.47.188 - - [26/Apr/2002:12:04:42 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 206 32768 "-"
129.110.47.188 - - [26/Apr/2002:12:04:43 -0700] "GET /103/LinkedListBasics.pdf
HTTP/1.1" 206 37760 "-"

```