Client-side State

Siddharth Kaza
Towson University, Computer and Information Sciences

Shopping cart

- *Assuming that you have done till Chapter 8 – creating the store catalog*
- Want to keep track of items that user has selected for possible purchase
- Need to track some "state" of the application
- Other examples
  - In session: Login credentials/authorization
  - Cross session:
    - Links to previous activity – purchases, posts (for blog), etc.
- Problem - HTTP is a "stateless" protocol
  - No inherent provision for maintaining state

Stateless HTTP

- Server does not retain any information between requests
- Essentially, each request is completely independent.
- Benefits of this design?
  - Speed
  - Simplicity
- Costs?
  - Security – no authentication in the protocol
  - Application does the heavy-lifting for the states

Ways to work around the lack of state

- URL
  - encoded parameters
  - http://www.foo.com/index.cgi?userid=2935
  - foo.cgi?key=value&key=value&key=value...
  - Easy to fake out – not at all secure.
  - Type of URL rewriting

- Hidden form field
  - Encode some information in a hidden field on a form
  - <input type="hidden" name="userid" value="2935">
  - Not much better than URL-encoding

Problems with these methods

- Hard to bookmark
- Problems with caches
  - Might maintain multiple copies since URL is different
- Server configuration and overhead
  - Parsing etc.
- Need to save information in stable storage
  - Everything will need to be stored server-side, even small bits of transient information

Cookies

- Client-side data for tracking web sessions
- Server asks for client to set a cookie
  - Name
  - Attribute value pairs
  - Max-age
  - Comments
  - Other fields
  - May request no cache
- Client sends a cookie in the HTTP request header
  - Domain, port, secure or not , attribute value pairs, etc.

Cookies (cont.)

- Client can reject cookies
  - Particularly if they present security concerns
    - Path doesn't match request path
- Cookies of same name overwrite
- Discard older cookies
- When a request is sent to a server, cookies for that host and URI are sent back.
- Name and other information encoded in cookie can be used to identify user, etc..
  - Only need 1 good identifier

Cookies and controversy

- Initially very controversial
- Let sites store data on client machines? No way?
  - Who would control?
  - What would they store?
  - What would it be used for?

How to use

- Avoid problems with cookies by specifying proposed "best practices"
- Use cookies when
  - User is aware
  - User can delete at any time
- Session information should not contain sensitive information
  - SSN would be a bad cookie attribute

How to use

- Inappropriate usage of cookies
  - Don't leak info. about users to other parties without explicit consent
  - Cookies are not an authentication mechanism
    - Just because you've got my cookie doesn't mean you're me.

- Web clients
  - MUST NOT respond to cookie requests unless explicitly enabled by the user
  - SHOULD provide interface for review of cookie requests
  - SHOULD provide interface for disabling of state to a server.

Cookies in RAILS

- **Cookies – store small amounts of data directly in user's** browser

- Special controller attribute cookies – hash

```
class CookiesController < ApplicationController
    def index
            cookies[:the_time] = Time.now.to_s;
            redirect_to :action => "time"
    end
    def next_visit
            cookie_value = cookies[:the_time]
            render(:text => "The last time you visited was #{cookie_value}")
    end
end
```

More complex cookie

```
cookies[:the_name] = { :value = > “Web",
   :expires = > 30.days.from_now
   :path => “/cookies”}
```

The browser will only send the cookie back
If the path matches

- Re-examine cookie contents
  - The default cookie (without these options) expires when the browser closes

Cookies in J2EE

- Putting a cookie into a response to a client
  - "response" is the object that forms response to a request

```
Cookie myCookie = new Cookie("myCookie",
   "someValue");
myCookie.setMaxAge(86400);

// Add cookie to the response
response.addCookie(myCookie);
```

- from http://www.ibm.com/developerworks/java/library/j-roadmap1/

Cookies in J2EE

- Receiving a cookie from a client

```
Cookies[] myCookies = request.getCookies();
// The getCookies method will return null if there
  are no cookies

if (myCookies != null) {
  for (int i = 0; i < myCookies.length; i++) {
      Cookie eachCookie = myCookies[i];
      // Do something w/each cookie
      ....
  }
}
```

- from http://www.ibm.com/developerworks/java/library/j-roadmap1/

Cookies in .net

- Write 2 cookies (C#)

```
Response.Cookies["userName"].Value = "patrick";
Response.Cookies["userName"].Expires =
      DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("lastVisit");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

- http://msdn2.microsoft.com/en-us/library/ms178194.aspx

Cookies in .net

- Read cookies

```
System.Text.StringBuilder output = new
       System.Text.StringBuilder();

HttpCookie aCookie;
for(int i=0; i<Request.Cookies.Count; i++)
{
  aCookie = Request.Cookies[i];
  output.Append("Cookie name = " +
       Server.HtmlEncode(aCookie.Name)
       + "<br />");
  output.Append("Cookie value = " +
       Server.HtmlEncode(aCookie.Value)
       + "<br /><br />");
}
Label1.Text = output.ToString();
```

- http://msdn2.microsoft.com/en-us/library/ms178194.aspx

Comparison

- .NET, J2EE very similar
- Powerful, but manual
- Rails – simpler, more constrained at first.

problems with cookies?

- Can't store lots of data
  - All gets re-transmitted each time
- Plain text – can be seen in the clear in browser
- Solution – use sessions
  - A session is a cookie that holds key to database entry
  - If you need to access multiple database records, you can always have additional cookies that are database keys
    - For example, a cookie for login status, a cookie for shopping cart
- Strategy: use cookies to hold Ids of relevant data on server.

Rails sessions (section 20.3)

- session – hash-like structure associated with a controller
  - Store key-value pairs during processing of a request
  - These pairs are available during subsequent requests

- Where is the session stored?
  - All options have their pros and cons
  - Client side – in the cookie store (browser) - default
    - PRO – simple, scalable, CONS - insecure
  - Temp file on the server,
    - PRO – simple, CON – Doesn't scale to multiple servers
  - In a table on the database
    - PRO – distributed, CON – slow(er)
  - In memory
    - CON – dumped when it crashes or the server runs out of memory?
  - DRB (distributed ruby servers)
    - CON – complicated, PRO - fast

Session example

- All session stores use a cookie to store a unique ID for each session (you must use a cookie, Rails will not allow you to pass the session ID in the URL as this is less secure).
- All Rails applications will have a session, whether you use it or not
- By default, the session is rails is stored on the client side in the cookie store
  - The data is cryptographically signed to make it tamper-proof.
- config/initializers/session_store.rb tells us where things go
- Same code as with cookies, but session instead.

Session example (cont.)

```
Code on blackboard
```

Lets look at how this is stored in the browser.

Contents of cookies

Name: jamssessionsession_id

Content: 1cf1f6b101f13e29dceee65fa11566a5

- This is the time, but, it is hashed using a MD5 function. The key is in the config/initializers/session_store.rb file
- What is the problem with this approach? Are we handling any of the problems with cookies?
  - Size issues?
    - No, the 4K limit still applies
  - Transfer time issues?
    - No, you are still transferring all the data with each request
  - Security issues?
    - Well, some, the data is hashed. But make sure the key is good and long.

Solution: Sessions – in the DB

- Uncomment in config/initializers/session_store.rb

```
config.action_controller.session_store =
  :active_record_store
```

Cleaning up sessions

- Problem with server-side sessions is that you keep accumulating them on the server.
  - Don't hold this data forever.
  - Sessions expire, people go away
- Its also a security issue – as a bank, you want to expire the session as soon as the user stops being active.
- File system solution
  - Cron job to delete temp session files.
- DB – repeated query

```
delete from sessions
where now( ) - updated_at > 3600;
```

Downsides to sessions

- Some folks don't like cookies (though this is a limited audience)
- Browsers break
- Firewalls can block cookies
- Do both? URL-encoding and/or hidden forms along with cookies?

Passing error messages between views - Flash

- The flash is a special part of the session which is cleared with each request.
  - This means that values stored there will only be available in the next request, which is useful for passing error messages etc.

- For instance, on a user logging out

```
class LoginsController < ApplicationController
  def destroy
    session[:current_user_id] = nil
    flash[:notice] = "You have successfully logged out."
    redirect_to root_url
  end
end
```

- http://guides.rubyonrails.org/action_controller_overview.html

Passing error messages between views – Flash (cont.)

- The *destroy* action redirects to the application's root_url, where the message will be displayed (code)
- For instance, in the view associated with the root_url

```
<html>
  <!-- <head/> -->
  <body>
    <% flash.each do |name, msg| -%>
      <%= content_tag :div, msg, class: name %>
    <% end -%>

    <!-- more content -->
  </body>
</html>
```

- http://guides.rubyonrails.org/action_controller_overview.html