Maximizing Aicache benefits: Client-Side Personalization Processing.

Introduction.

Many a website uses a customary "greeting" section in their web pages, showing something to the effect of "***Hello Mr.Smith***" or "***Welcome smith11***" when a *logged-in* user is detected. When an *anonymous* visitor opens these pages, a login form is displayed instead, with customary "username" and "password" fields.

Frequently, the user name or login id text (such as ***Mr.Smith*** or ***smith11***) within the "greeting" message is a link , such as *userprofile.php* or *profile.asp,* to user settings screen where user preferences can be changed, messages read etc.

Some site don't require visitors to register for all or most of their content (major news sites are like that), some only offer small subset of content to anonymous users and require registration and login in order to the see the "premium" content.

Most of the time such "greeting" section is built via server-side code (PHP, Java, ASP etc). When login form is first submitted and after successful authentication of the user, a new HTTP session is created and *username*, *userid* and some other information might get stored in it. As such logged-in user then visits different pages on the site, every time the page is generated a new and the appropriate greeting gets incorporated into the page. The session is frequently associated with a session cookie. Some common session cookie names include "JSESSIONID", "PHPSESSID" etc.

The problem statement.

Outside of the greeting message, the rest of the web pages look the same for all visitors - both logged-in registered user and anonymous visitors alike. Yet a piece of code might be executed every time pages are rendered (served), inserting a personalized greeting message into the pages, making them non cacheable - all at a regular price of stressing most of your infrastructure - from web servers to application code to DB servers. It also complicates the setup, as you need to assure the session state gets replicated so all of the application code can see it or a provision is made for sticky connections.

However, there's an easy to eliminate such bottlenecks and enable caching of content- by moving "greeting" processing logic to client side ! Just a few lines of JavaScript and viola : you can accomplish 95% caching ratios while personalizing content still ! Eat your cake and still have it, here's how.

Client-Side Solution.

The provided example a small html page that showcases an example of client-side greeting logic processing. There're no changes to your server-side code if your login logic generates a cookie that we can use as *userid* or *username* in the greeting. If no such cookie is being generated, it is trivial to have one implemented.

We assume that we can use value of a cookie called "*username*" in the greeting section. So, if *username* is set to *smith11,* the greeting will read: "Hello smith11" . We don't HREF (link) username to a URL (such as "editprofile.jsp") to save space, yet it is trivial to do so. You can also have links to regular "Profile", "Messages" added as part of the same code.

The code (in red) simply looks to see if a username cookie is set and if it is, it replaces the login form with the greeting. As simple as that. Users still have to login as usual to have the cookie set - no changes there. When a logged-in user desires to go to a profile page, the regular code will still execute to make sure the user is logged in (that one normally relies on a session cookie). No compromises in security.

Here's the code. Copy and save it in a file of your choosing. The code relies on *prototype.js* (you can obtain this very commonly used JS library at http://www.prototypejs.org/) and Cookie convenience code you can obtain at http://www.red91.com/2008/02/04/cookies-persistence-javascript.

---------- Copy from here ----------

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html><head><meta http-equiv="Content-type" content="text/html; charset=utf-8">

<title>Blank Page</title>

<script src="prototype.js" type="text/javascript"></script>

</head><body><h1>Client-Side Greeting/Login processing.</h1>

<div id="bucket"><form action="login.php?action=login" method="post" id="login">

Username: <input type="text" name="user"/><br/>

Password: <input type="password" name="password"/><br/>

<a id="sign_in" onclick="login.submit()" title="Sign in">Sign in</a>

<a id="sign_up" href="/registration.php" title="Register.">Register</a>

<a id="forgot" href="/passreminder.php" title="Reminder">Reminder</a><br/>

</form></div>

<div class="news-item" id="item_1">

<h3>Declaration of Independence</h3>

<p id="summary_1">When, in the course of human events ...</p>

</div>

<script type="text/javascript">

<!--

if (Cookie.get('username') != null ) {  $('bucket').update('<h3> Hello Mr.' + Cookie.get('username') + '<\h3>'); }

 //-->

</script></body></html>
```

---------- To here ----------

Testing.

To test, we will use Firebug, the de-facto golden standard of all of the JS developers in the world. Assuming you have Firebug installed in a recent Firefox browser, simply open the file you created.

As no *username* cookie is presently set, you will see a login form on top of the page, with common login form attributes.

Let's set the *username* cookie and see what happens then. Click on Firebug icon and get to the Firebug console.

With the page open, type at Firebug's console prompt (>>>):

Cookie.set('username','maximus')

We have set a session cookie with the name of *username* and value of *maximus*. To verify that it is now set, you can type :

Cookie.get('username')

You should see the *'maximus'* value displayed, signifying the presence of the cookie.

Refresh the page. You will see that login form is gone and instead, an appropriate user greeting is displayed. Mission accomplished ! Now, this page and any other page that relies on client-side personalization, can be safely cached , with all of the numerous benefits of caching, yet it will be personalized via a friendly greeting that users expect to see after they log-in.

Conclusion.

We have shown, via a simple example, how to move personalization logic to client-side, enabling page caching. There's nothing stopping us from extending this example to include more evolved personalization. Things like checking for messages (via XMLHTTPRequest, for example) and displaying the "*you have NNN messages*" link, showing a low balance warnings, hiding or showing links to premium content, etc .

The benefits of caching be truly a life saver for many a Web site - a difference between staying up and being down, snippy response times and unacceptably long waits, 10 servers vs. 100 servers, $50,000 vs. a million dollar spend on your infrastructure, life or death ... well, that was an exaggeration, but you get the point.