Subversion, More MVC pattern, Caching

CS174

Chris Pollett

Nov. 17, 2008.

Outline

- Subversion
- Caching
- More MVC Pattern

Seeing who changed what

- We have already seen how to generate a diff between trunk and a local copy using svn diff.
- To see a log of changes that have been made on the repository, one can use a line like:

  svn log #in the local copy directory one wants to #see changes for.

- You can also add -v for verbose and -r to single out revision numbers
- To check out or update to old copies of the repository you can use svn co -rXXX or svn up -rXXX
- To see line by line who wrote the line you can use svn blame.

SVN and HTTP

- We mentioned that SVN can be configured to run over HTTPS, we briefly mention some of what involved in setting this up.
- First, you need to edit the httpd.conf file so that the following two modules are loaded:

LoadModule dav_module modules/mod_dav.so

LoadModule dav_svn_module modules/mod_dav_svn.so

- Next we need to tell Apache that if the url path begins with /repos/ then use svn:

```
<Location /repos>
  DAV svn
  SVNPath /var/svn/repository
</Location>
```

SVN Permissions

- The easiest/dumbest way now to authenticate svn is to use the basic authentication that we used previously with .htacces files.
- So we would add to our <Location> directive lines like:

```
AuthType Basic
 AuthName "Subversion repository"
 AuthUserFile /etc/svn-auth-file
Require valid-user
```

And use htpasswd to add/modify passwords in that file.

- It is probably slightly safer to send hashes of passwords over the net, in which case rather than basic authentication one should use Digest:

```
AuthType Digest
 AuthName "Subversion repository"
 AuthDigestDomain /svn/
 AuthUserFile /etc/svn-auth-file
 Require valid-user
```

SSL and SVN

- The above set-up completely works from the server perspective if https rather than http is being used.
- The svn client might ask you if you want to accept the servers certificate.
- It is also possible that the server is set up to ask for a .pem file from the client (which might be cached for later use).
- Often you will want to allow public reading of the repository but only authenticated, writing. This can be achieved by using the <LimitExcept> tag within your <Location> directive:

```
<LimitExcept GET PROPFIND OPTIONS REPORT>
   Require valid-user
  </LimitExcept>
```

Caching

- Svn is an example of a service that we might be running on our website that facilitates managing our code.
- In addition to it, so far we have a web-server and a database server.
- There are several other servers that we might use on our site associated with caching.
- One use for caching is to avoid database reads, by caching objects read from the database.
- In order to understand how this works, we need to look at our MVC set-up again for our website.

The MVC Pattern

- So far we have split our web app into the following units:
  - A main index page, which loads a config page. This page also decides which controller should be used.
  - We have a directory of controllers each having its own class, extending some base class, each controller might be used to handle different kinds of requests: admin, landing, post message, etc
  - We have a directory of models one model handles each mapping from database tables of a specific kind of object. Again, models are classes each extending some common base class
  - Views -- these are mainly html pages which render our site. Which view to use is chosen by the controller.

Other Common Units in a Web Site

- We might have in our website:
  - A directory with all our css files
  - A directory with all our js files
  - A directory of third party apps used
  - A directory of elements -- these are portions of web-pages that might be used across multiple views
  - A directory of helpers -- these are classes of functions which are designed to make it easier to output certain UI objects like select tags, etc.
  - A directory of components -- these are shared between controllers and implement certain common functionality not involving a database model.

Object Relational Mappings

- Typically, models are used to get objects into and out of a database.
- As an example, a user might made many posts to a discussion board.
- This is an example of a one-to-many relationship
- We might have two database tables user and post. To represent this relationship the post table might have a column uid as a foreign key reference into the user table.
- Now a User object might be modeled as PHP associative array consisting of the name, and other fields of the user, one of these fields being an array of posts objects:

  Array("name" => "Bob", …, "posts" => Array([0] => Array("title" => "Dark and Stormy Night", …), [1]=>…))
- An object relational mapping is a mapping between these kind of objects and database tables.
- Typically, besides one-to-many relationship, one might need to handle many-to-many, belongs-To, and one-to-one relationships.
- In pretty much all cases, marshalling an object requires several database queries, which can be slow, so may be sped up by caching.

Memcached

- Memcached (http://www.danga.com/memcached/) allows you to create an in memory cache of data.
- A memcache can be on a separate server from either your database or your webserver.
- Your web app connects to the memcache daemon and checks if an item such as an object that might have been obtained from a database using an ORM mapping, is in the memcache. If it is you can get them quickly, if it isn't then you go to the database.
- In PHP you can access a memcache daemon using the PECL extension: http://pecl.php.net/package/memcache

Using memcached

- To run the memcached daemon one can type a line like:

memcached -d -m 50 -l 127.0.0.1 -p 11211

Here: -d -- as daemon

-m -- number of megabytes

-l -- location

-p -- port

- To use memcached within PHP one needs to create a memcache object:

  $memcache = new Memcache();

- Next one needs to set-up the memcache server:

  $memcache->addServer($host, $port, $persistent, $weight, $timeout, $retry_interval);

- To add something to the cache one could use a command like:

  $memcache->set($key, $val, $flag, $expire);

  // $flag is whether to compress the item in the store

  //$expire - Unix timestamp, or seconds from present. 0 -don't expire

- To read from the cache one can use a line like:

  $memcache->get($key);

Trivial Memcache Script

```
<html>
<head><title>Memcache Test</title></head>
<body>
<h1>Test</h1>
<?php
  $memcache = new Memcache();
  $memcache->addServer("localhost", 11211, true, 1, 1, 5);
  $memcache->set("bob", "value", null, 60);
  echo "<p>".$memcache->get("bob")."</p>";
?>
</body>
</html>
```