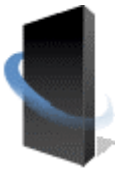
Zend Platform™

Technical Article:

Zend Platform's Partial Page Caching



By **Zend Technologies**

September 2005

Zend
The *php* Company

Zend Platform's Partial Page Caching

Real-World Examples

One of the most powerful features of Zend Platform is the ability to boost your application's performance by caching the output of your scripts. While full-page caching is good in certain situations, in many cased you will need to have parts of your page fully-dynamic, while other parts of the page are static or do not change each time the page is requested. For such cases, Zend Platform has the partial page caching API - php functions that give you the power to enhance your application's performance dramatically and still have fully dynamic content. In this article I am going to give some examples on how to smartly implement partial page caching on real-world php applications.

Data caching vs. output caching

Zend's caching API provides us with two basic methods of caching:
data caching and **output caching**.

Data Caching

Data caching takes the return value of a specified built-in or user-defined function and caches it as a string. By using the *serialize* and *unserialize* functions you can also cache arrays and objects. You should not use data caching on functions that generate actual output - when the code is cached it will not be executed and you will have no output generated. The functions used for data caching are *output_cache_fetch*, *output_cache_put* and *output_cache_get*. You can find the full description of these functions in Zend Platform's user guide.

Output Caching

Instead of caching the return value of your php code, output caching will work on the actual output of your code. It is relatively easy to wrap segments of code that generate HTML (or other) output with output caching control functions, thus preventing the need to execute the code every time. These functions are *output_cache_exists* and *output_cache_stop*. You can also use the function *output_cache_output*, which works in a manner resembling data caching - it will cache the output of a specified function for a specified period of time.

On top of those, another important caching API function is *output_cache_remove_key* which allows you to clear a specific cache segment from within the code itself, regardless of its lifetime.

What caching is not?

Keep in mind that partial page caching, as powerful as it is, is simply caching. It will take the output or data generated by your code and cache it. The aim is to minimize the amount of executed code over a long period of time, and to make sure code is only executed when we need to generate (ore re-generate) dynamic content. It will not optimize your code (that is make your code more efficient) and will not accelerate your code (that is make your code run faster) - it will simply skip the compilation and execution of your code unless it is necessary.

Best-practices in implementing partial content caching

In a perfect world, you would have designed your application from scratch with Zend's API in mind, implementing partial content caching and other Zend Platform APIs as you code. While this would be best, the situation in most cases is that people have an up-and-running application before they think about implementing Zend's performance features. In such cases, you will need to find the places in your code that take long time to execute, or that are executed often, and should be cached. Then you can consider how to cache them, for how long, and whether to use data caching or output caching. While this is application-specific, these few pointers might help:

- Try to locate output-generating loops and wrap them with output caching functions. Common examples for loops that generate content might be <select> menus for choosing dates, or a list of site news that is retrieved from a database.
- Database access is usually a major time consumer. In many cases you can use data caching minimize database queries - but keep in mind that database queries return resource pointers, not the actual data. You need to convert your result set to an array or object before you can serialize and cache it.
- You can use a variable in the ID of the cached segment - this allows you to create different cached versions of the same code, for example for each different value your function's parameter might have.
- Try to optimize caching by defining a reasonable lifetime for each cached section. A 5 day weather forecast might be updated every 6 hours, while a current weather report might be updated every 5 minutes.
- Try to locate content and data that is shared between scripts and users. In many cases this can replace the usage of include files.

Implementation examples

In the following examples, I will try to demonstrate an implementation of various caching techniques on existing code. A good sample application is a php-based forum system. Since the aim is to keep this article generic (and not too long), only necessary code will be written, so you can understand the application logic. You can find the full description of these API functions and others in Zend Platform's user guide, as well as additional examples.

Consider the following function:

```
<?php
  // Get an array containing all the forums that we have and the number of
messages in each forum
  function listForums()
  {
    // Query DB and get list of forums
    $result = mysql_query("....");

    // Generate an array from the result and return it
    $forums = array();
    while ($row = mysql_fetch_array($result))
    {
      $forums[] = $row;
    }

    return $forums;
  }
?>
```

Here we have a function that generates a two-dimensional array with all the forums that we have and some information about each forum. We use these functions in various places in our code and on different pages, for example in the main index page as well as on the search page in a dropdown menu. Since we want to use the same data in different presentation implementations, the best practice here would be to use data caching. If for example, I have used the following code in my search page:

```
<?php
  $forums = listForums();
  put
  // Generate a <select> tag from $forums
  foreach ($forums as $forum)
  {
    echo "...";
  }
?>
```

I can simply replace the line setting $forums with this line:

```
<?php
  $forums = unserialize(output_cache_fetch("ForumsList",
"serialize(listForums())", 600));
?> put
```

output_cache_fetch will check if "ForumsList" is a valid cache ID. If not, it will run the code in the second parameter and return it's value. If it is valid, it will simply fetch the value cached under this ID and return it, skipping the execution of the code. This way we save the need to access the database each time the page is called. More then that, I can use the same cached data in my main index page and in all my other pages, since I'm only caching the return value of listForums() and not the loop that generates the actual HTML output.

On most web forums, you would get much more visitors that are just reading or browsing through posts that actually posting. It would simply be a waste not to cache the threads in my forum as static HTML. This can be done quite easily. If I have a script named topic.php which is responsible for displaying a specific topic, and gets a $_GET parameter 'topic_id' which holds the specific thread ID, I simply use two API functions to cache this thread:

```
<?php
  $topic_id = $_GET['topic_id'];
  // Check if this topic is aleady cached.
  // If it does - print it's contents. If not - execute the code.
  if (! output_cache_exists("ViewTopic$topic_id", 3600))
  {

    // ...
    // Part of the code that prints out the thead...
    // ...

    output_cache_stop();      // Stop caching here
  }
?>
```

Notice that we have wrapped our HTML generating code with two simple functions. We have also used the unique thread ID in the cache ID - so each thread has it's own cached version. Some of you may wonder whether one hour (3600 seconds) it not a very long time to cache a forum thread. Most of us will want the thread to update the second someone posts a reply, right? No problem. All we have to do here is to add the following line at the code section that handles new posts:

```php
<?php
  // ...
  // Receive reply and store it in the database ...
  // ...

  // Clear cache for a specific unique ID
  output_cache_remove_key("ViewTopic$topic_id");
?>
```

This way I ensure that the cache is being refreshed when a topic is updated. Again, I have used the topic number in the cache ID.

The sky is the limit!

These examples are, of course, a very small part of what you can achieve. While they can give you a view on what can be done with Zend Platform's partial page caching API, implementations are very different from application to application. The key element here is creativity - In contrast to Code Acceleration that works "out of the box", partial caching is up to you to set up. The more time and brainpower you are willing to invest into using it smartly, the more performance you gain.

Contact Information

United States and Canada:
Zend Technologies, Inc.
19200 Stevens Creek Blvd.
Cupertino, CA 95014
Tel: 1-888-PHP-ZEND (1-888-747-9363)
Fax: 1-408-253-8801

Central & Eastern Europe:
Zend Technologies GmbH
Bayerstraße 83
80335 München, Deutschland
Tel: +49-89-516199-0
Fax: +49-89-516199-20

International:
Zend Technologies, Ltd.
7 Abba Hillel Street
Silver Building
Ramat Gan, Israel 52136
Tel: 972-3-613-9665
Fax: 972-3-613-9671

Document Feedback:
E-mail feedback to:
documentation@zend.com
Please state the name of the
document in the subject line.