Getting to Know the WordPress Transients API

Key Takeaways:

- *Caching* is a key process for speeding up website performance. It saves difficult-to-retrieve data (such as extensive calculations or complex database queries), saves them, and serves the saved values rather than running the calculations again.
- The *Transients API* is WordPress's solution for *object caching*: caching individual pieces of data rather than whole webpages. The Transients API allows developers to cache objects freely, without worrying about the specific tools available on a given web server.
- Use of the transients API is relatively simple, with two main functions: `get_transient()` and `set_transient()`. Each transient is a piece of data with a specific expiration date.

Computer performance revolves, in large part, around the idea of *caching*: "storing something in a more-ready and quicker-to-access state," so that you can more quickly deliver the final result.

To take a real-life example, your kitchen is a kind of cache! You've got soup cans "cached" in your cupboard, ready for quick delivery to your stove—meaning that you don't have to drive to the supermarket every time you want soup.

The WordPress Transients API is a tool for caching, and an important way to improve performance in WordPress.

Why to Use Transients

In a WordPress context, caching most often means "full page" caching: storing a full webpage just before it's sent out to a visitor, so that the next visitor to ask for that page will get the stored version without your server having to rebuild it. This is the approach of many caching plugins, such as WP Super Cache. In some situations this is ideal, and can have a major impact on site speed.

> You can improve performance by caching things other than full pages, such as slow results from remote servers like Facebook's, or large database queries.

However, there's also the idea of "partial" or "object" caching: achieving performance gains by storing things other than full pages, such as the results of long-running computations, slow results from remote servers (such as Facebook or Twitter results), or large database queries which are likely to be both slow to run and very consistent in the results they yield.

In WordPress, the way to "partially cache" pages, one data object at a time, is the Transients API.

Understanding the WordPress Transients API

The WordPress Transients API creates its own, easy-to-use interface for the various means and methods of caching data in a given server environment, so that WordPress developers don't have to worry too much about the specifics of that environment.

With Transients, we want to be able to name and store a hunk of data, and to get it back quickly. This so-called "key-value store" is exactly what *in-memory caching systems* allow. Memcached (memcached.org) is one of these systems: a very fast, simple, and powerful way to store blobs of data and retrieve them later. However, Memcached and similar systems are not available everywhere—in fact, not even on *most* servers where WordPress runs.

> The Transients API lets developers cache data as if Memcached or a similar key-value storage system is available—whether or not it actually is.

So the Transients API lets WordPress programmers cache data just like they would if Memcached or a similar key-value storage system were available—without needing to worry whether it actually is.

In-Memory Storage and In-Database Storage

As a note on how the Transients API actually works: when Memcached or other forms of caching aren't available, WordPress stores the cached data in the `options` table of its regular database, to provide the same functionality as an in-memory cache—although, unfortunately, somewhat slower.

Transients Should Be Transient!

These "caches" or "transients" aren't meant to be permanent. (Permanent data should live in the Options API; see *Mastering the Options API in WordPress*.)

When we cache data, we want to set a defined expiration time for that data, after which it'll simply disappear.

How to use the Transients API

The Transients API is pretty simple: you first store a name-value pair, and then you retrieve it.

`set_transient()`

Here's what setting a transient looks like:

```
$string = "Cache me for a day!";
$bool_response = set_transient( 'wpshout_cache_me', $string, 86400 );
```

Here, we're using `set_transient()` to store a transient named `wpshout_cache_me`. This transient will have a value given by `set_transient()`'s second argument. In this case, that's the value of `$string`: the string "Cache me for a day!"

How long will this transient persist? That's our third argument, which takes an integer number of seconds. `wpshout_cache_me` will persist for 86,400 seconds—that is, for one full day.

`get_transient()`

Retrieving a transient for use looks like this:

```
$transient_string = get_transient( 'wpshout_cache_me' );

if ( false === $transient_string ) {
    return; // In real life we'd want to set_transient() here
}

echo '<h1>' . $transient_string . '</h1>';
```

Here, we're using `get_transient()`, with the transient's name (`wpshout_cache_me`) as its only parameter, to retrieve the transient if it exists.

Be careful! `get_transient()` will return `false` if the transient doesn't exist. So it's really important to test for the transient's existence before using it. That's what our `if`-statement does. In real code, the lack of a transient would be our excuse to `set_transient()` all over again, but we've omitted that here.

If we have successfully retrieved the transient, you can do anything with it. In this case, we're printing it out, wrapped in an `<h1>` tag.

That's it! There are four other functions: `delete_transient()`, which manually clears transients out of the cache, and three alternative functions for use in WordPress Multisite. But there's plenty here to get you, as it were, "up and running" with transients.

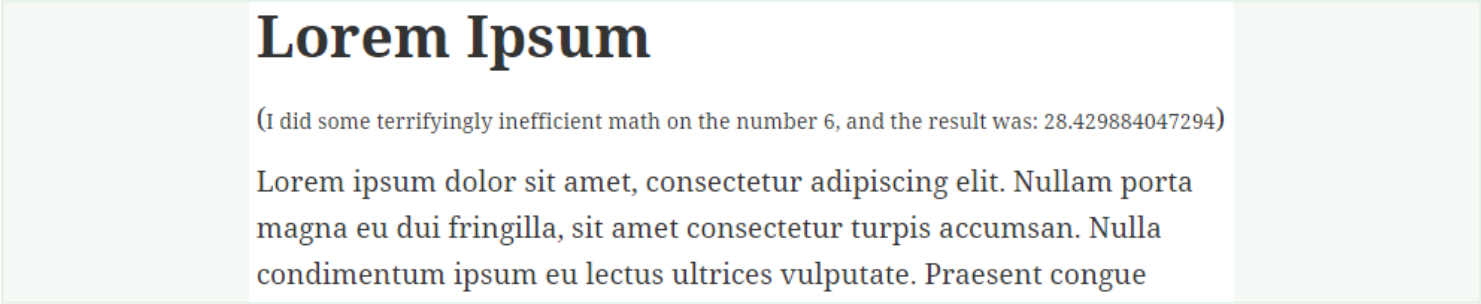
An Example: Fun With Transients

Let's look at how transients work, in a full-fledged example called "WPShout Cache Hard Math."

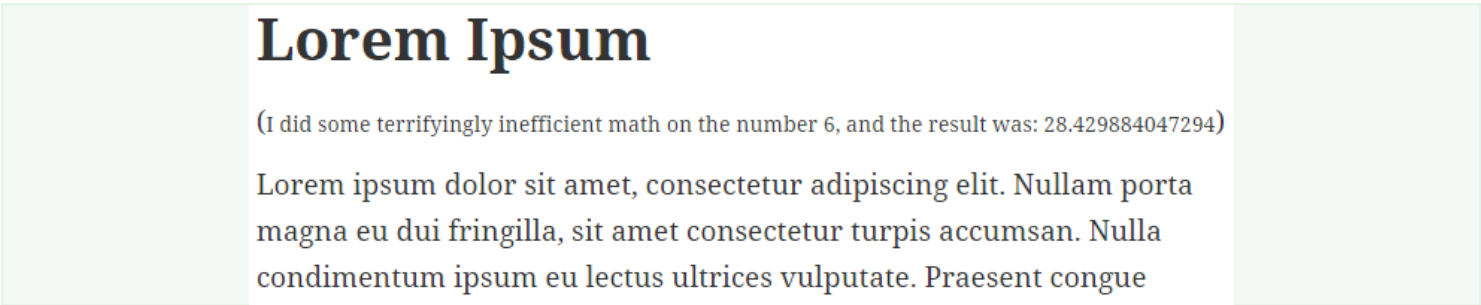
How the Finished Product Works

Our plugin is a demo of object caching: it tells the server to do millions of complex calculations, and then to cache the results for ten seconds. So every ten seconds, the site loads painfully slowly —but in the seconds between, it loads quickly! That's caching at work.

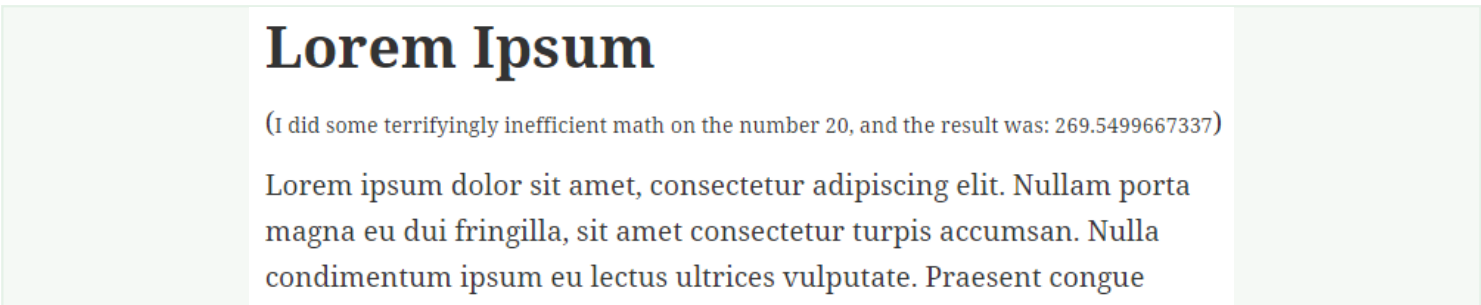
(Note: since it intentionally hurts site performance for demo purposes, this isn't a plugin you'll want to deploy on your own sites, or on anyone's site that you'd like to stay on good terms with.)



The result at six seconds past the minute. This page took around ten seconds to load!



The same result, at twelve seconds past the minute. This page loaded quickly.



The result at twenty seconds past the minute. This page loaded very slowly again.

The Code

We're about to show you a whole plugin file, in chunks. Each chunk comes immediately after the previous chunk, so if you copy-paste them all you'll have a working—but silly and not-to-be-deployed—plugin.

This section does a huge amount of calculations, and `returns` an array with two elements: the input value before the calculations, and the output value after them:

```
<?php
/*
Plugin Name: WPShout Cache Hard Math
*/

function wpshout_do_hard_math( $int ) {
    // $start is the starting integer
    $start = $int;

    // Insanely processing-intensive calculations
    $i = 0;
    while($i < 100000) {
        $int = pow(sqrt(sqrt(sqrt(sqrt($int)))),16.0001);
        $i++;
    }

    // Return our array: what we started with and what resulted
    return array ( $start, $int );
}
```

This section attempts to get the transient that is the result of the `wpshout_do_hard_math()` calculations. If it finds there's no transient, it will try to set the transient, then get it. It then `return`s either the transient, or `false` if getting the transient failed:

```
function wpshout_get_hard_math_transient() {
    // Get the transient
    $result = get_transient( 'hard_math' );

    if ( false !== $result ) {
        // Transient exists, so return it
        return $result;
    }

    // Get array from doing "hard math" (on seconds elapsed in current minute)
    $mathed = wpshout_do_hard_math( date( 's' ) );

    // Attempt to set transient with array results; timeout is 10 seconds
    $bool_response = set_transient( 'hard_math', $mathed, 10 );

    if( false === $bool_response ) {
        // Setting the transient didn't work, so return false for failure
        return false;
    }

    // Transient is now set, so get it and return it
    return get_transient( 'hard_math' );
}
```

This section attempts to retrieve the transient. If it succeeds, it hooks into `the_content` to print a string containing the transient's calculations at the top of the post's content:

```
function wpshout_filter_content_with_hard_math_transient( $content ) {
    // Get the transient
    $result = wpshout_get_hard_math_transient();

    // If transient isn't an array, just return content unaltered
    if ( ! is_array( $result ) ) {
        return $content;
    }

    // Prepend string with transient data to content and return it
    return '(I did some terrifyingly inefficient math on the number ' . ltrim(
$result[0], '0' ) . ', and the result was: ' . $result[1] . ')' . $content;
}
add_filter('the_content', 'wpshout_filter_content_with_hard_math_transient');
```

Notes on This Example

If you understood this example right away, you're really getting WordPress at a deep level! If not, don't worry too much about the specifics—just try to absorb the basic uses of `get_transient()` and `set_transient()` that this example exists to show off.

One More Practical Use Case for Transients

In case you're having trouble picturing how transients could *benefit* a site (rather than make it virtually unusable), we'll link to a very beautiful real-world example of caching a nav menu, which includes a use of `delete_transient()` to invalidate the cache when the nav menu changes: http://leaves-and-love.net/transients-speed-up-wordpress-theme/.

What We've Learned about WordPress Transients

We've learned how and why to use transients in WordPress, and gotten deep into an example that shows them at work. We're a lot better-equipped to cache expensive operations in WordPress—and our sites will be faster for it. Onward!

Summary Limerick

If your site is so slow that it's crashing,
You should probably look into caching—
For whole pages, it's true,
But for smaller things, too!
With transients, caching is smashing.

Quiz Time!

1. All transients should be:
    A. Stored in the `wp_options` table
    B. Intended to expire
    C. The results of processing-intensive database queries

2. If `get_transient()` fails to find the transient it's looking for, it will `return`:
    A. `false`
    B. A PHP error
    C. The most recent value of the transient

3. Once retrieved, a transient is:
    A. Simply another piece of site data
    B. A piece of site data subject to expiration
    C. A piece of site data subject to expiration, and given lower priority than other data

Answers and Explanations

1. `B.` `A` and `C` are both sometimes true, but `B` is the reason why transients exist as a separate concept from site options.

2. `A.` This means that you always have to check that your `get_transient()` calls returned something other than `false` before using the results.

3. `A.` Once retrieved, a transient's value is used identically to any other piece of data. It's not "subject to expiration" once already retrieved; if the transient had expired, it could not be retrieved. As such, `B` is false. `C` is gibberish.