← WordPress.com for Windows Metro and the REST API

Nginx, SPDY, and Automattic →

Querying Posts Without query_posts

Posted on 2012-05-14 07:03:14 by John James Jacoby

Here at WordPress.com, we have over 200 themes (and even more plugins) running inside the biggest WordPress installation around (that we know of anyway!) With all of that code churning around our over 2,000 servers worldwide, there's one particular WordPress function that we actually try to shy away from; `query_posts()`

If you think you need to use it, there is most likely a better approach. query_posts() doesn't do what most of us probably think it does.

We think that it:

- Resets the main query loop.
- Resets the main post global.

But it actually:

- Creates a new WP_Query object with whatever parameters you set.
- Replaces the existing main query loop with a new one (that is no longer the main query)

Confused yet? It's okay if you are, thousands of others are, too.

This is what query_posts actually looks like:

```
/**
 * Set up The Loop with query parameters.
 *
 * This will override the current WordPress Loop and shouldn't be used
 * once. This must not be used within the WordPress Loop.
 *
 * @since 1.5.0
 * @uses $wp_query
 *
 * @param string $query
 * @return array List of posts
 */
function &query_posts($query) {
    unset($GLOBALS['wp_query']);
    $GLOBALS['wp_query'] = new WP_Query();
    return $GLOBALS['wp_query']->query($query);
}
```

Rarely, if ever, should anyone need to do this. The most commonly used scenario is a theme that has featured posts that appear visually before the main content area. Below is a screen-grab of the iTheme2 theme for reference.

Recent Posts

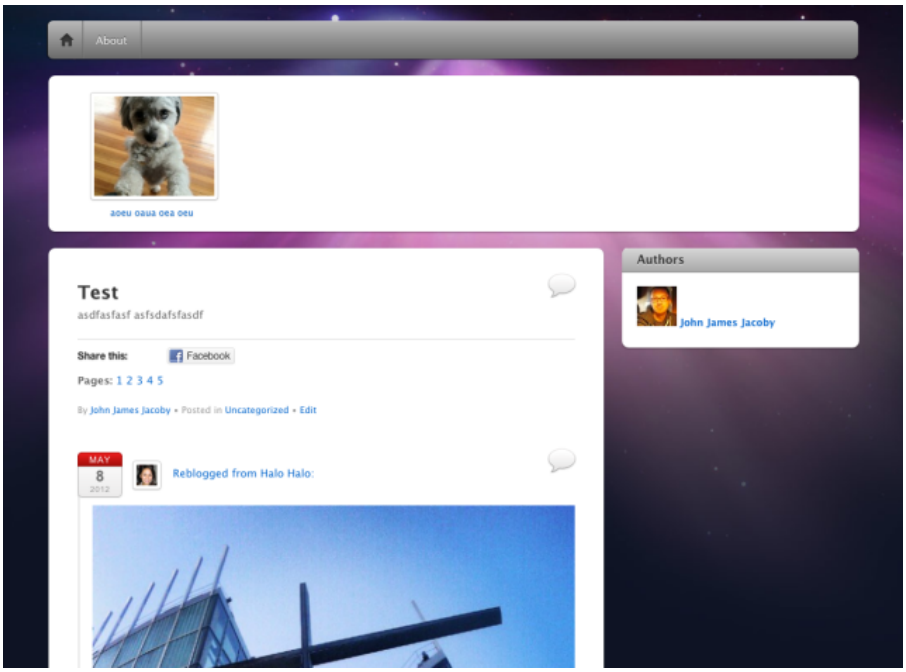
- Custom post type and metadata support in the REST API
- Developer plugin 1.2 released: UI Improvements and New Plugins
- Meet Justin Shreve
- Meet Beau Lebens
- Platform Updates: Posting Endpoints

Archives

- April 2013
- February 2013
- December 2012
- November 2012
- October 2012
- August 2012
- June 2012
- May 2012
- April 2012

Categories

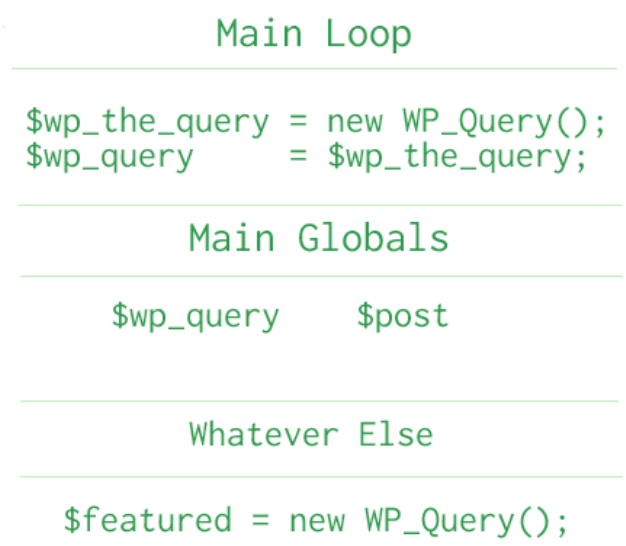
- APIs
- Interviews
- Lessons
- News
- Platform Updates
- Plugins
- Uncategorized



The thing to keep in mind, is by the time the theme is starting to display the featured posts, WordPress has already:

- looked at the URL…
- parsed out what posts fit the pattern…
- retrieved those posts from the database (or cache)…
- Filled the $wp_query and $post globals in PHP.

Let's think about it like this:



The "Main Loop" consists of 3 globals, 2 of which actually matter.

- $wp_the_query (does not matter)
- $wp_query (matters)
- $post (matters)

The reason $wp_the_query doesn't matter is because you'll *never* directly touch it, nor should you try. It's designed to be the default main query regardless of how poisoned the $wp_query and $post globals might become.

Back to Featured Posts

When you want to query the database to get those featured posts, we all know it's time to make a new WP_Query and loop through them, like so…

```php
$featured_args = array(
    'post__in' => get_option( 'sticky_posts' ),
    'post_status' => 'publish',
    'no_found_rows' => true
);

// The Featured Posts query.
$featured = new WP_Query( $featured_args );

// Proceed only if published posts with thumbnails exist
if ( $featured->have_posts() ) {
    while ( $featured->have_posts() ) {
        $featured->the_post();
        if ( has_post_thumbnail( $featured->post->ID ) ) {
            /// do stuff here
        }
    }

    // Reset the post data
    wp_reset_postdata();
}
```

Great! Two queries, no conflicts; all is right in the world. You are remembering to use wp_reset_postdata(), right? If not, the reason you do it is because every new WP_Query replaces the $post global with whatever iteration of whatever loop you just ran. If you don't reset it, you might end up with $post data from your featured posts query, in your main loop query. Yuck.

Remember query_posts()? Look at it again; it's replacing $wp_query and not looking back to $wp_the_query to do it. Lame, right? It just takes whatever parameters you passed it and assumes it's exactly what you want.

I'll let you stew on that for a second; let's keep going…

What if, after your featured-posts query is done and you've dumped out all your featured posts, you want to *exclude* any featured posts from your main loop?

Think about this…

It makes sense that you would want to use query_posts() and replace the main $wp_query loop, right? I mean, how else would you know what to exclude, if you didn't run the featured posts query BEFORE the main loop query happened?

EXACTLY!

Paradox, and WordPress and WP_Query are designed to handle this extremely gracefully with an action called '[pre_get_posts](#)'

Think of it as the way to convince WordPress that what it wants to do, maybe isn't really what it wants to do. In our case, rather than querying for posts a THIRD time (main loop, featured posts, query_posts() to exclude) we can modify the main query ahead of time, exclude what we don't want, and run the featured query as usual. Genius!

This is how we're doing it now in the iTheme2 theme:

```php
/**
 * Filter the home page posts, and remove any featured post ID's from :
 * onto the 'pre_get_posts' action, this changes the parameters of the
 * before it gets any posts.
 *
 * @global array $featured_post_id
 * @param WP_Query $query
 * @return WP_Query Possibly modified WP_query
 */
function itheme2_home_posts( $query = false ) {

    // Bail if not home, not a query, not main query, or no featured p
    if ( ! is_home() || ! is_a( $query, 'WP_Query' ) || ! $query->is_m
        return;

    // Exclude featured posts from the main query
    $query->set( 'post__not_in', itheme2_featuring_posts() );

    // Note the we aren't returning anything.
    // 'pre_get_posts' is a byref action; we're modifying the query di
```

```
}
add_action( 'pre_get_posts', 'itheme2_home_posts' );

/**
 * Test to see if any posts meet our conditions for featuring posts.
 * Current conditions are:
 *
 * - sticky posts
 * - with featured thumbnails
 *
 * We store the results of the loop in a transient, to prevent running
 * extra query on every page load. The results are an array of post ID
 * match the result above. This gives us a quick way to loop through fe
 * posts again later without needing to query additional times later.
 */
function itheme2_featuring_posts() {
    if ( false === ( $featured_post_ids = get_transient( 'featured_post

        // Proceed only if sticky posts exist.
        if ( get_option( 'sticky_posts' ) ) {

            $featured_args = array(
                'post__in'      => get_option( 'sticky_posts' ),
                'post_status'   => 'publish',
                'no_found_rows' => true
            );

            // The Featured Posts query.
            $featured = new WP_Query( $featured_args );

            // Proceed only if published posts with thumbnails exist
            if ( $featured->have_posts() ) {
                while ( $featured->have_posts() ) {
                    $featured->the_post();
                    if ( has_post_thumbnail( $featured->post->ID ) ) {
                        $featured_post_ids[] = $featured->post->ID;
                    }
                }

                set_transient( 'featured_post_ids', $featured_post_ids
            }
        }
    }

    // Return the post ID's, either from the cache, or from the loop
    return $featured_post_ids;
}
```

It reads like this:

- Filter the main query.
- Only proceed if we're on the home page.
- Only proceed if our query isn't somehow messed up.
- Only proceed if we want to filter the main query.
- Only proceed if we actually have featured posts.
- Featured posts? Let's check for stickies.
- Query for posts if they exist
- (At this point, WP_Query runs again, and so does our 'pre_get_posts' filter. Thanks to our checks above, our query for featured posts won't get polluted by our need to exclude things.
- Take each post ID we get, and store them in an array.
- Save that array as a transient so we don't keep doing this on each page load.
- We're done with featured posts, and back in our main query filter again.
- In our main query, exclude the post ID's we just got.
- Return the modified main query variables.
- Let WordPress handle the rest.

With a little foresight into what we want to do, we're able to architect ourselves a nice bit of logic to avoid creating a third, potentially costly WP_Query object.

Another, more simple example

The Depo Masthead theme wants to limit the home page to only 3 posts. We already learned earlier we *don't* want to run query_posts() since it will create a new WP_Query object we don't need. So, what do we do?

```php
/**
 * Modify home query to only show 3 posts
 *
 * @param WP_Query $query
 * @return WP_Query
 */
function depo_limit_home_posts_per_page( $query = '' ) {

    // Bail if not home, not a query, not main query, or no featured p
    if ( ! is_home() || ! is_a( $query, 'WP_Query' ) || ! $query->is_ma
        return;

    // Home only gets 3 posts
    $query->set( 'posts_per_page', 3 );
}
add_action( 'pre_get_posts', 'depo_limit_home_posts_per_page' );
```

Stop me if you've heard this one. We hook onto 'pre_get_posts' and return a modified query! Woo woo!

Themes are the most common culprit, but they aren't alone. More often than not, we all forget to clean up after ourselves, reset posts and queries when we're done, etc… By avoiding `query_posts()` all together, we can be confident our code is behaving the way we intended, and that it's playing nicely with the plugins and themes we're running too.

This entry was posted in Lessons and tagged query_posts, WordPress, WP_Query. Bookmark the permalink.

← WordPress.com for Windows Metro and the REST API

Nginx, SPDY, and Automattic →

24 thoughts on "Querying Posts Without query_posts"

William P. Davis (@williampd) says:

2012-05-14 15:54:07 at 15:54
Any objections to using get_posts instead of WP_Query?

Reply

John James Jacoby says:

2013-03-26 00:43:18 at 00:43
Not at all; get_posts() actually uses WP_Query directly.

Reply

Daniel Bachhuber says:

2012-05-16 20:35:50 at 20:35
Reblogged this on danielbachhuber.

Reply

Joey Kudish says:

2012-05-16 21:32:02 at 21:32
Reblogged this on Joachim Kudish and commented:
Really good explanation of why you shouldn't use query_posts; a function I know I've used in the past but definitely stay away from now 😀

Reply

Jeremy says:

2012-05-17 12:21:36 at 12:21
Reblogged this on Jeremy Herve.

Reply

1. Pingback: Austin WordCamp and Developer Day, 2012 | TTI COM Visual Media
2. Pingback: Querying Posts Without query_posts - Konstantin Kovshenin

Ben says:

2012-06-06 21:37:30 at 21:37
Hi John – This sounds very clever but to be honest I just don't understand what you're suggesting so am hoping you can explain a bit further?

I totally understand the use of WP_Query and no problem with not using query_posts. I also understand the theory of using the filter before the initial query is called – however I don't understand how this reduces the amount of queries or why it's better than using multiple WP_Queries?

From what I understood above – you're still doing 2 WP_queries – so how is this better?

Reply

John James Jacoby says:

2013-03-26 00:45:32 at 00:45
It's better in that you're only querying for exactly the posts you want, rather than trying to merge posts and loops together, hoping to remove duplicates or intersections.

Reply

Steveorevo says:

2012-06-06 23:19:44 at 23:19
Awesome. That clears up a lot! Question: are you forgetting to wp_reset_post_data() after line 49's new WP_Query in your iTheme2 theme code example? Per your initial suggestion; just wanted clarification.

Reply

Juana Maria Bauza says:

2012-06-14 17:12:24 at 17:12
Sorry for my dummy question but… Where do I add this code?
…and to change the main query only for the RSS Feed, where do I add this code?

Thanks

Reply

Stephen Cronin says:

2012-06-30 12:27:13 at 12:27
Am I missing something here, or will that transient never expire? I think you need to specify an expiry parameter so it will refresh every so often and pick up changes to sticky posts etc…

Reply

tonilehtimaki says:

2012-10-25 10:12:45 at 10:12
I was thinking exactly the same.

Reply

WraithKenny says:

2012-07-05 18:20:15 at 18:20

Awesome, looks a lot like Andrew Nacin's WordCamp presentation but is easier to read then the slides. Thanks John!

Reply

WraithKenny says:

2012-07-05 21:02:57 at 21:02
PS. might be worth throwing in a is_admin() check next to ! is_main_query() else it'll effect your edit.php I think.

Reply

Bjarni Wark says:

2012-07-17 00:09:20 at 00:09
Thanks for the explanation and clarification on the usage of query_posts. Very helpful and good to be more in the know of best practices with WordPress.

Reply

Luis Diego says:

2012-07-19 16:47:39 at 16:47
You can also chage the query within the "request" filter..
http://codex.wordpress.org/Plugin_API/Filter_Reference/request
In case you need an advance control!

Reply

John James Jacoby says:

2013-03-26 00:48:57 at 00:48
Good point. The 'request' filter may actually be more accurate, in that you can skip the is_main_query() check. Though, you'll likely want to replace it with an is_admin() check, so it's probably a wash.

Reply

Eddie Moya says:

2012-08-11 00:42:21 at 00:42
You said query_posts() doesnt look back, and with WP_Query you can use wp_reset_postdata() – but isnt that the same thing as using wp_reset_query() after query_posts()?

I always thought new WP_Query(); …. wp_reset_postdata() was functionally identical to query_posts();…. wp_reset_query() – with the only real difference being that your creating a new objet with one, and modifying the original object with the latter. I was under the impression wp_the_query somehow handled this properly in either case. Am I incorrect?

Reply

Paul Maitland says:

2012-12-20 12:21:55 at 12:21
I was thinking the exact same thing Eddie. Granted my tech knowledge of WordPress is a tad limited, but using query_posts seems to be the more straight forward and direct method (at least to me).

Reply

John James Jacoby says:

2013-03-26 00:57:03 at 00:57
Good questions, and I'll try to address them all.

You're correct that $wp_the_query and wp_reset_query(), do handle the query_posts() usage correctly. wp_reset_query(), is_main_query(), and query_posts(), are all designed to help developers manipulate the main query request.

In the first example above, once the transient is populated, the second query (of three possible queries) will not occur until the transient is deleted, and the original query will still happen as WordPress intended for it to, minus the post ID's from the transient. If not for caching the post ID's in the transient, three queries would happen:

* WordPress does the main request query.
* We request the sticky posts.
* We query_posts() minus the sticky posts.

By planning a little bit ahead, we eliminate 1 query from the process.

Reply

3. Pingback: How to do damn near anything with WordPress – Stephanie Leary

samhudi suharjo ibnu hasyim says:

2012-12-20 01:10:12 at 01:10
thanks

Reply

4. Pingback: Jeremy Herve | Querying Posts Without query_posts

Leave a Reply

Enter your comment here...

An Creation