CACHING AND SCALING WITH FRAGMENT CACHING

Erick Hitter (@ethitter)

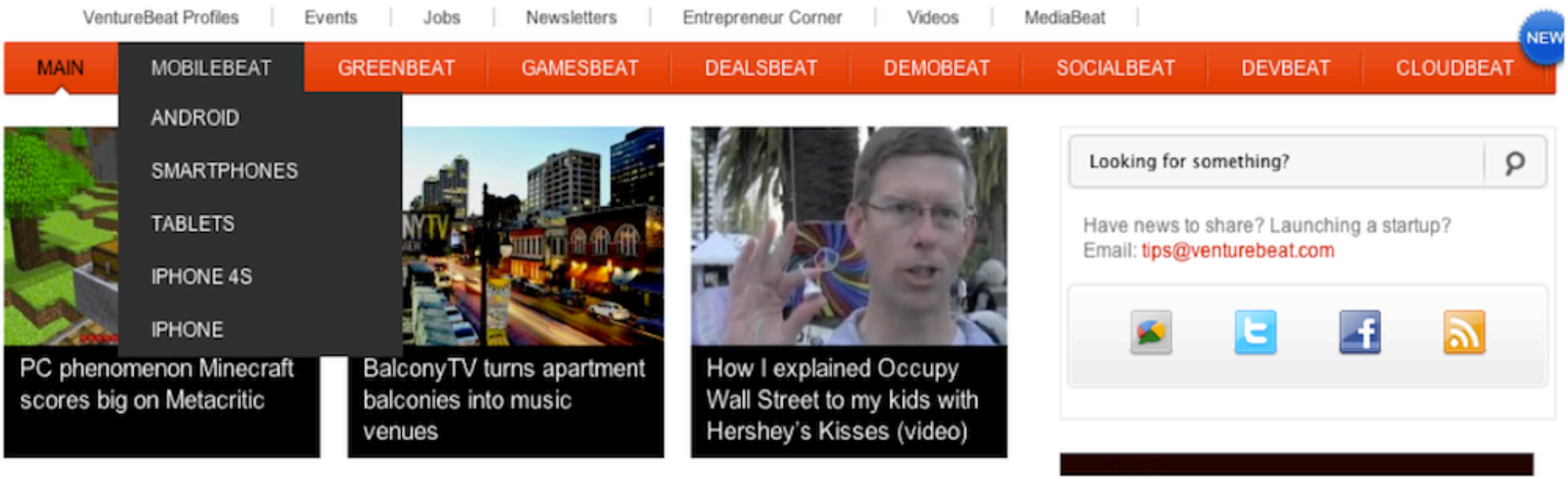
Lead WordPress Developer at Oomph, Inc.
WordPress 3.3 Core Contributor
WordCamp Boston Organizer
Plugin Author

WHAT IS CACHING?

- Serving static content to a visitor rather than something generated upon request.
- Two major types in WordPress context (there are more):
  - Page - whole page is static and, therefore, could be outdated.
  - Fragment - cache pieces of a page rather than the entire page.

FRAGMENT CACHING BENEFITS

Allow dynamic and static content to coexist



FRAGMENT CACHING BENEFITS

Common elements can be reused throughout a site

About MobileBeat

VentureBeat's Mobile channel focuses on news about everything mobile, including smartphones, feature phones and tablets. We cover mobile phones, mobile operating systems, mobile ads, and the burgeoning mobile app ecosystem. Major topics include the iPhone, Android, iPad, BlackBerry, Palm webOS, Windows Phone 7 and Symbian. Some of the companies we cover includes Apple, Google, Microsoft, RIM, Palm, Nokia, Samsung, HTC and Motorola.

About GreenBeat

GreenBeat covers the intersection of innovation and investment in green technology. It's where we report on everything from startups to broader trends in the billion-dollar cleantech market — a wide range of subjects that include solar, wind, biofuels, electric cars, energy efficiency technologies and all aspects of the smart grid, including smart meters, smart grid infrastructure, networking, data management and demand response.

VB Writers


Chikodi Chima
Writer


Dean Takahashi
Lead Writer, GamesBeat


Devindra Hardawar
Senior Editor, MobileBeat Lead


Dylan Tweney
Executive Editor


Heather Kelly
Senior Editor


Jennifer Van Grove
Writer


Jolie O'Dell
Writer


Matt Marshall
Founder & Editor-in-Chief


Meghan Kelly
Writer


Sean Ludwig
Writer


Tom Cheredar
Writer


Ciara Byrne
Contributor

VB Writers

Chikodi Chima
Writer

Dean Takahashi
Lead Writer, GamesBeat

Devindra Hardawar
Senior Editor, MobileBeat Lead

Dylan Tweney
Executive Editor

Heather Kelly
Senior Editor

Jennifer Van Grove
Writer

Jolie O'Dell
Writer

Matt Marshall
Founder & Editor-in-Chief

Meghan Kelly
Writer

Sean Ludwig
Writer

Tom Cheredar
Writer

Ciara Byrne
Contributor

FRAGMENT CACHING BENEFITS

Reduce calls to APIs

WORDPRESS' NATIVE CACHING APIS

Transients

- Persistent out of the box
- Stored in wp_options: _transient_*{key}*
- WordPress uses for certain internal functions
- *set_, get_, and delete_transient()*

Object Cache

- Not persistent without a plugin, such as W3 Total Cache or Memcached Object Cache
- Storage depends on server's and plugin's capabilities
- Used extensively within WordPress
- Cache objects can be grouped
- *wp_cache_add(), _set, _get, _delete*

FRAGMENT CACHING BASICS: CREATING

```
<?php
        function generate_cached_output() {
                if ( false === ( $output = wp_cache_get( $cache_key, $cache_group ) )
) {
                        $output = 'Something to be cached';

                        wp_cache_set( $cache_key, $output, $cache_group, 86400 );
                }

                return $output;
        }
?>
```

- wp_cache_get() returns a boolean false if the requested cache isn't set or has expired.
- Everything that is to be cached must be accessible via a function that returns its results rather than echoing them, otherwise output buffering is needed.
- wp_cache_add() will not overwrite an existing, unexpired cache, whereas wp_cache_set() does.

FRAGMENT CACHING BASICS: CLEARING

```
<?php
        function clear_cached_output( $new_status, $old_status ) {
                if ( 'publish' == $new_status || 'publish' == $old_status )
                        wp_cache_delete( $cache_key, $cache_group );
        }
        add_action( 'transition_post_status', 'clear_cached_output', 10, 2 );
?>
```

This above example clears a cache when anything is published or something that is published is modified. The "something" could be a post, page, or custom post type object.

If, instead, the cache should be rebuilt only when posts are edited, one additional argument from *transition_post_status* can be employed.

```
<?php
        function clear_cached_output( $new_status, $old_status, $post ) {
                if ( ( 'publish' == $new_status || 'publish' == $old_status ) &&
'post' == $post->post_type )
                        wp_cache_delete( $cache_key, $cache_group );
        }
        add_action( 'transition_post_status', 'clear_cached_output', 10, 3 );
?>
```

FRAGMENT CACHING BASICS: CLEARING

Same cache generation function from two slides ago, with a minor change

```
<?php
        function generate_cached_output( $force_refresh = false ) {
                if ( $force_refresh || false === ( $output = wp_cache_get(
$cache_key, $cache_group ) ) ) {
                        $output = 'Something to be cached';

                        wp_cache_set( $cache_key, $output, $cache_group, 86400 );
                }

                return $output;
        }
?>
```

Clear by rebuilding cache

```
<?php
        function clear_cached_output( $new_status, $old_status ) {
                if ( 'publish' == $new_status || 'publish' == $old_status )
                        generate_cached_output( true );
        }
        add_action( 'transition_post_status', 'clear_cached_output', 10, 2 );
?>
```

UNPREDICTABLE KEYS

- Object caching doesn’t provide a way to clear all caches in a given group.
- Therefore, if the cache key is unpredictable, how can we clear it?
- For example, a list of recent posts to be displayed on an individual post, but that excludes the current post.

UNPREDICTABLE KEYS: RECENT POSTS

```
<?php
        function recent_posts( $post_id = false, $qty = 3 ) {
                $post_id = (int) $post_id;
                if ( ! $post_id )
                        return false;

                $qty = (int) $qty ? (int) $qty : 3;

                $cache_key = $post_id . '_' . $qty;

                if ( false === ( $output = wp_cache_get( $cache_key, 'recent_posts' )
) ) {
                        $output = 'Something to be cached';

                        wp_cache_set( $cache_key, $output, 'recent_posts', 86400 );
                }

                return $output;
        }
?>
```

UNPREDICTABLE KEYS: CACHED ARRAY

```
<?php
        function recent_posts( $post_id = false, $qty = 3 ) {
                /* Sanitize function arguments */

                $cache_key = $post_id . '_' . $qty;

                $cache = wp_cache_get( 'single', 'recent_posts' );
                if( ! is_array( $cache ) )
                        $cache = array();

                if ( ! array_key_exists( $cache_key, $cache ) ) {
                        $output = 'Something to be cached';

                        $cache[ $cache_key ] = $output;
                        wp_cache_set( 'single', $cache, 'recent_posts', 86400 );
                }

                return $output;
        }
?>
```

UNPREDICTABLE KEYS: CACHED ARRAY

Pros

- Cache can easily be cleared because a single object with a predictable key is set.
- Cache is only rebuilt if specific post ID/quantity key is absent from array.
- Better for cache elements that are reliably small.
- Allows for checking existence of various keys, such as in a loop.

Cons

- Object caching configuration may limit size of individual cache objects.
- Array corruption could invalidate an entire cache object.
- Array can become bloated if different quantities are used simultaneously.

UNPREDICTABLE KEYS: INCREMENTOR

```
<?php
        function get_cache_incrementor() {
                $incrementor = wp_cache_get( 'incrementor', 'recent_posts' );
                if ( ! is_numeric( $incrementor ) ) {
                        $incrementor = time();
                        wp_cache_set( 'incrementor', $incrementor, 'recent_posts',
86400 );
                }

                return $incrementor;
        }

        function recent_posts( $post_id = false, $qty = 3 ) {
                /* Sanitize function arguments */

                $cache_key = get_cache_incrementor() . '_' . $post_id . '_' . $qty;

                if ( false === ( $output = wp_cache_get( $cache_key, 'recent_posts' )
) ) {
                        $output = 'Something to be cached';

                        wp_cache_set( $cache_key, $output, 'recent_posts', 86400 );
                }

                return $output;
        }
?>
```

WHERE WE USE FRAGMENT CACHING

- Custom loops
- Anytime a lot of data must be retrieved from WordPress and parsed.
- Most situations where WP_Query generates a subquery outside of the main query.
- Almost anything that is reused across multiple pages.

CATEGORY__NOT_IN VS POST__NOT_IN

```
<?php
        new WP_Query( array(
                'category__not_in' => 167
        ) );
?>
```

```
SELECT … WHERE 1=1  AND wp_posts.ID NOT IN ( SELECT tr.object_id FROM
wp_term_relationships AS tr INNER JOIN wp_term_taxonomy AS tt ON tr.term_taxonomy_id
= tt.term_taxonomy_id WHERE tt.taxonomy = 'category' AND tt.term_id IN ('167') ) …
```

CATEGORY__NOT_IN VS POST__NOT_IN

```
<?php
        function cached_get_objects_in_term( $term_ids, $taxonomies, $args ) {
                /* Sanitize function arguments */

                $cache_key = md5( implode( ',', $term_ids ) . $taxonomies .
serialize( $args ) );

                if ( false === ( $ids = wp_cache_get( $cache_key,
'get_objects_in_term' ) ) ) {
                        $ids = get_objects_in_term( $term_ids, $taxonomies, $args );
                        /* Error check $ids */
                        wp_cache_set( $cache_key, $ids, 'get_objects_in_term', 86400
);
                }

                return $ids;
        }
?>
```

CATEGORY__NOT_IN VS POST__NOT_IN

```
<?php
        new WP_Query( array(
                'post__not_in' => cached_get_objects_in_term( 167, 'category' )
        ) );
?>
```

Before

```
SELECT … WHERE 1=1  AND wp_posts.ID NOT IN ( SELECT tr.object_id FROM
wp_term_relationships AS tr INNER JOIN wp_term_taxonomy AS tt ON tr.term_taxonomy_id
= tt.term_taxonomy_id WHERE tt.taxonomy = 'category' AND tt.term_id IN ('167') ) …
```

After

```
SELECT … WHERE 1=1  AND wp_posts.ID NOT IN ( '1','2','3','4','5' ) …
```

MENU CACHING: NO ACTIVE STATES

```
<?php
	function cache_wp_nav_menu( $args = false, $skip_cache = false ) {
		/* Sanitize function arguments */

		$echo = (bool) $args[ 'echo' ];
		$args[ 'echo' ] = false;

		$cache_key = md5( implode( '|', $args ) );

		if( $skip_cache || false === ( $menu = wp_cache_get( $cache_key,
$this->cache_group ) ) ) {
			$menu = wp_nav_menu( $args );

			wp_cache_set( $cache_key, $menu, $this->cache_group, 86400 );
		}

		if( $echo )
			echo $menu;
		else
			return $menu;
	}
?>
```

MENU CACHING: ACTIVE STATES

```php
<?php
        function cache_wp_nav_menu( $args = false, $skip_cache = false ) {
                /* Sanitize function arguments */

                $echo = (bool) $args[ 'echo' ];
                $args[ 'echo' ] = false;

                $queried_object = (int) get_queried_object_id();
                if ( ! $queried_object )
                        $queried_object = 0;

                $cache_key = get_cache_incrementor();
                $cache_key .= $queried_object . '|';
                $cache_key .= implode( '|', $args );
                $cache_key = md5( $cache_key );

                ...
?>
```

MENU CACHING: KEYS & CLEARING

- *get_queried_object_id()* returns an integer representing the post ID or term ID.
- Front page and custom post type archives return *0*.
- Menu caches must be cleared when four different actions fire to ensure consistency:
  - wp_update_nav_menu
  - wp_update_nav_menu_item
  - wp_delete_nav_menu
  - wp_setup_nav_menu_item

QUERY_POSTS() VS PRE_GET_POSTS

query_posts()

- Function provided to modify main query.
- Runs after main query has already executed.

```
<?php
        //In home.php
        query_posts( array(
                'posts_per_page' => 5
        ) );
?>
```

pre_get_posts

- Action used to modify any query.
- Runs before every query executes.

```
<?php
        function action_pre_get_posts(
$query ) {
                if ( $query->is_home() )
                        $query->set(
'posts_per_page', 5 );
        }
        add_action( 'pre_get_posts',
'action_pre_get_posts' );
?>
```

HOW DOES THIS FACTOR INTO OUR WORK ON WORDPRESS.COM VIP?

- Page caches only last for five minutes.
- No page caching for logged-in users.
- Sites that publish with great frequency trigger regular invalidations of homepage and category pages.
- Web servers outnumber database servers.

Want to know more about WordPress.com infrastructure? Check out http://goo.gl/lYpJH.

QUESTIONS?

Email: erick@thinkoomph.com

On Twitter: @ethitter

Slides will be available at http://www.ethitter.com/.