The Beginner's Guide to WordPress Actions and Filters

Tom McFarlin on Oct 17th 2012 with 23 Comments

Tutorial Details

-
- **Topics:** Hooks: Actions, and Filters
- **Difficulty:** Beginner
- **Estimated Time:** 1 hour

When it comes to professional WordPress development, it's imperative that developers understand both actions and filters – that is, it's important to understand WordPress hooks.

Simply put, hooks are what give us the ability to customize, extend, and enhance WordPress through an API in our themes, plugins, and other custom development efforts.

The problem is that these two features of WordPress – arguably the most important aspects of developing for the platform – are either widely misunderstood or completely ignored.

In this post, we're going to take a look at the WordPress page life cycle, understand how hooks work, and review the differences in actions and filters so that we may not only become better theme and/or plugin developers, but also have a deeper understanding of how WordPress works.

Note: This article is targeted specifically for beginners, so if you're an experienced developer this may be a bit of a refresher; however, if you *are* a beginner, please feel free to leave questions in the comment area at the end of the post.

The WordPress Page Lifecycle

Before we actually begin talking about the WordPress hooks, it's important to understand how the WordPress page lifecycle works.

> A page life cycle is nothing more than a combination of the events that take place from when a browser requests a page to when the server returns the page to the browser.

Let's say, for example, that you're loading up a single page. Then, at a high level, WordPress will do something like the following:

- Look at the requested page ID
- Query the database for the page by its ID
- Query the database for any associated data (such as categories, tags, images, etc)
- Query the database for the comments associated with it
- Return *all* of the data to the browser

The template files and the calls to the API functions are then responsible for rendering, styling, and positioning the data on the screen.

It sounds simple, but if you think about some of the most complex blogs that you read, or if you think even about some of the work that you've done, you can begin to understand just how intensive this particular process can be.

Of course, this is at the most simplistic level, too. This doesn't any include any caching mechanisms or any of the advanced topics that others often discuss when building WordPress-based projects.

A popular WordPress developer Rarst – the guy behind queryposts.com – has put together a relatively detailed graphic that shows the WordPress core load:

Do *not* be discouraged if you can't follow the above diagram. I've placed it here simply as a reference. The ultimate goal of this session is that, by the end, all developers are able to understand it.

With that said, here is the key thing to understand about hooks during the WordPress page load lifecycle:

> While WordPress is running its series of queries and preparing to render data back to the browser, it's looking at all of the custom hooks – that is, the actions and filters – that have been written and is passing data through those filters before returning data to the browser.

At this point, it's important to define hooks and look at the differences between actions and filters and how they play into this whole life cycle.

All About Hooks

WordPress hooks refer to two things – actions and filters. If you were to look at the Codex articles on hooks, you'd see nothing but a short page linking to the references for both actions and filters. That's exactly how it should be too, because *that's* what hooks are.

So here's how to think about this:

> Hooks enable us to literally hook into parts of the WordPress page lifecycle to retrieve, insert, or modify data, or they allow us to take certain actions behind the scenes.

Pretty cool, right?

But there are two things to understand about this:

1. Actions are different than Filters.
2. You can't simply throw a hook into an arbitrary point of execution. There are times during which certain hooks fire and optimal times for you to take advantage of that.

That said, let's first define actions and filters, then we'll review when to fire what.

Taking Action

So, what are actions? The simplest definition is this: Actions indicate that something has happened. That's it. But what good is that definition? And how do we not get that confused with events?

Here's how I keep it straight:

> Actions are events in the WordPress page lifecycle when certain things have occurred – certain resources are loaded, certain facilities are available, and, depending on how early the action has occurred, some things have yet to load.

Since we've discussed the WordPress page lifeycle, actions are basically certain points during the life in which you can introduce your own functionality.

This means that you have the ability to make something happen while a page is loading.

The Codex provides a *great* resource on the actions that are built into WordPress as well as the order in which they fire. ***Bookmark, refer often, and learn this.***

It is fundamental in learning how to hook into WordPress at certain points during it's execution and Doing It Right™ .

Case in point: I often see developers hooking into the `init` action far too frequently. Sure, there's a time for doing this. But say that you wanted to do something just before getting the posts. Then it'd make sense to use the `pre_get_posts` hook rather than `init`, right?

That's why it's important to understand actions.

Filter All The Things

Filters, on the other hand, are completely different to actions. Like actions, they are similar points that occur during the WordPress page lifecycle; however, what they *do* is different.

Here's how I define filters:

> Filters are functions that WordPress passes data through during certain points of the page lifecycle. They are primarily responsible for intercepting, managing, and returning data before rendering it to

> the browser or saving data from the browser to the database.

Assume for a moment that a site visitor is about to load a post. From what we understand of the WordPress page lifecycle, WordPress is going to query the database for that post, then return it to the browser. Before doing that, though, it's going to run the data through any filters that have been established.

At this point, the filters will take action on data that's being passed to them. For example, say you wanted to append a short sentence about the author at the end of the content. To do this, you'd register a custom function with `the_content` filter and append your sentence to the content.

How to do this is beyond the scope of this article, but we aim to cover this in a future article in the session.

Just as with actions, the Codex features a comprehensive list of available filters. And similarly, ***Bookmark, refer often, and learn this.***

Once you've got a solid understanding of filters, then you can begin manipulating data retrieval and serialize by Doing It Right™ rather than circumventing the WordPress API . It provides a powerful, very easy way to do manipulate data.

That's why it's important to understand filters.

But When Do I…?

At this point, the inevitable question always arises.

> When do I use which hook?

And here's the advice I normally give:

- Use actions when you want to add something to the existing page such as stylesheets, JavaScript dependencies, or send an email when an event has happened.
- Use filters when you want to manipulate data coming out of the database prior to going to the browser, or coming from the browser prior to going into the database.

That's it! Easy enough, I hope.

Conclusions

At this point, I highly recommend reviewing the following resources:

- The Plugin API which also provides some great information that can be used in Theme Development, too.
- The Action Reference
- The Filter Reference

We're planning to continue running several posts in the session to accompany this article as well as Pippin's article on extensible plugins so be sure to stay tuned to this session for more information on hooks.

Tags: plugin developmentTheme Development

By Tom McFarlin

Tom is a self-employed developer who loves writing, building, and sharing WordPress-based projects. He's the lead developer at 8BIT and technical editor of WP Daily. You can follow him on Twitter.

Note: Want to add some source code? Type <pre><code> before it and </code></pre> after it. Find out more