Services & Drupal

Services & Drupal

Jonathan Daggerhart

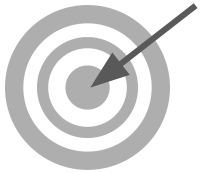
- **Developer at Hook 42**
- **Organizer for Drupal Camp Asheville**

Drupal.org: daggerhart
Twitter: @daggerhart
Blog: https://www.daggerhart.com

Drupal Camp Asheville
Site: https://drupalasheville.com
Twitter: @drupalasheville

Services & Drupal

What we will cover



1. What are services, and what problem do they solve?
2. Global Objects & Singleton Anti-Patterns
3. Service Locator
4. Drupal Services

Services - Overview

"Service" is a fancy word that roughly means -
"A class instance that is easily accessible and reusable by the system."

Problem: When building an application it's common to need some class method many times, and it is wasteful to re-instantiate the class.

A great example of this type of need in Drupal is in the formatting of timestamps as dates, leveraging the user-defined date formats.

Consider:
`/core/lib/Drupal/Core/Datetime/DateFormatter.php`

Services & Drupal

Global Objects & Singleton Anti-Patterns

One way to solve this problem of reusable class instances is with a global instance, or using the Singleton pattern.

But **neither of these are good solutions** because it creates tight coupling between the specific class and the application.

Example: Global object anti-pattern

```
global $date_formatter;

$date_formatter->format(1111111111, 'short');
```

Example: Singleton anti-pattern

```
$date_formatter = DateFormatter::getInstance();

$date_formatter->format(1111111111, 'short');
```

Services & Drupal

Example Service Locator

A better approach is to use the Service Locator pattern. This approach allows class instances to be registered and accessed with the system and results in much more loosely coupled code.

Service Locator stores instances by name

```php
class ServiceLocator {
  private $services = [];

  public function register($service_name, $instance) {
    $this->services[$service_name] = $instance;
  }

  public function get($service_name) {
    if (!empty($this->services[$service_name])) {
      return $this->services[$service_name];
    }

    return null;
  }
}
```

Retrieve an instance from the Service Locator by its name

```php
// Sometime during the system bootstrap the service is registered.
$service_locator = new ServiceLocator();
$service_locator->register('date.formatter', new DateFormatterService());

// Theoretically, some other part of the application could replace the
// service with a better implementation.
//$service_locator->register('date.formatter', new BetterDateFormatter());

// Later in our application, we can request the service by name.
$date_formatter_service = $service_locator->get('date.formatter');
print $date_formatter_service->format(1111111111);
print $date_formatter_service->format(1111111111, 'medium');
print $date_formatter_service->format(1111111111, 'long');
```

Services & Drupal

Services in Drupal 8 - Service Locator

Drupal has a Service Locator that is accessed by calling:
`\Drupal::service('service.name');`

The "`service.name`" is an arbitrary name used to identify the service. Some modules use a period between words, others use an underscore.

Retrieving common Drupal services:

```php
// Database object for custom queries.
$database = \Drupal::service('database');

// Date Formatter object for rendering timestamps.
$dateFormatter = \Drupal::service('date.formatter');

// Object for rendering Drupal render arrays.
$renderer = \Drupal::service('renderer');
```

Services & Drupal

Registering New Services w/ the Locator

Sometimes our module will want to register new services with Drupal's service locator so that other modules can make use of its functionality. To do this, we create a YAML file within our module.

`my_module/my_module.services.yml`

Arbitrary name, used to identify and retrieve the service.

Fully namespaced class.

```yaml
# Top level starts with "services:"
services:
  # Services are keyed by name that is used
  # to retrieve it.
  my.custom.service:
    # "class" property defines the class that
    # will be instantiated and returned.
    class: \Drupal\my_module\MyService
```

Services & Drupal

Retrieving Our Custom Service

After registering our service (and rebuilding the Drupal registry), we can retrieve and use it throughout the rest of the system.

`my_module/src/MyService.php`

```php
<?php

namespace Drupal\my_module;

class MyService {

  public function doSomethingRad() {
    return "Heck yeah! We did something.";
  }
}
```

`my_module/my_module.services.yml`

```yaml
# Top level starts with "services:"
services:
  # Services are keyed by name that is used
  # to retrieve it.
  my.custom.service:
    # "class" property defines the class that
    # will be instantiated and returned.
    class: \Drupal\my_module\MyService
```

`my_module/my_module.module`

```php
/**
 * Implements hook_help()
 */
function my_module_help() {

  $myService = \Drupal::service('my.custom.service');
  $myService->doSomethingRad();

}
```

Services & Drupal

Additional Service Definition Properties

When registering a service with Drupal there are a few additional properties that allow for more functionality. Notably the “arguments” and “tags” properties.

Arguments allow for dependencies to be injected into the service.
Syntax is like an array.

Tags identify the service to be of a special type. The system may use tagged services in a special way.
Syntax is an array of structures with the “name” property.

```
services:
  another.custom.service:
    class: \Drupal\my_module\AnotherService
    # Arguments allow for dependencies to be
    # injected into the service.
    arguments: ['@messenger']
    # Tags identify the service to be of a
    # special type. The system may use tagged
    # services in a special way.
    tags:
      - { name: 'event_subscriber' }
```