Services, Dependency Injection, and Containers, Oh my!

Jennifer Hodgdon (jhodgdon)

Pacific Northwest Drupal Summit

Portland, Oregon

October 18-19, 2014

What is a Service?

- Concept adopted from Symfony project
- Drupal functionality is built as a collection of “services”
- Each service is one chunk of the functionality (e.g., translating UI text, caching one type of data, URL path aliasing, etc.)
- Each service's functionality is provided by a PHP class
- ***The default class providing each service can be overridden by a module.***

Service Definitions

A service definition includes:

- A machine name. Examples: `string_translation`, `cache.default`, `path.alias_manager`
- An interface defining what the service does. This is implicit but a **VERY GOOD IDEA (TM)**
- A default class to provide the service
- Some types of services have tags
- Some have dependencies on other services
- Some have factories and other information

Defining a Service

Service definitions go in *.services.yml files. Examples from core/core.services.yml:

```
services:
  path.alias_manager:
    class: Drupal\Core\Path\AliasManager
    arguments: ['@path.alias_storage',
      '@path.alias_whitelist',
      '@language_manager', '@cache.data']
  cache.default:
    class: Drupal\Core\Cache\CacheBackendInterface
    tags:
      - { name: cache.bin }
    factory_method: get
    factory_service: cache_factory
    arguments: [default]
```

You'll also need to define the default class you specified, and an interface for your service.

Enabling Concept

Enabling concept for Services:

Obtain all service classes from the "Container," also known as the "Dependency Injection Container"

Two methods:

- Service Location (adequate)
- Dependency Injection (preferred)

Obtaining Services: Service Location

```
// Get any service, from outside a class:
$service = \Drupal::service('service_name');
// Equivalent, if you have a $container variable:
$service = $container->get('service_name');

// Some particular services:
$cache = \Drupal::cache();
$lock = \Drupal::lock()->acquire('mylock');
$query = \Drupal::entityQuery('myentity');
$config = \Drupal::config('mymodule.settings');
```

Obtaining Services: Dependency Injection

- Dependency injection is used in classes.
- If your class is a service, inject dependencies via `arguments` in the `*.services.yml` file. They'll get magically passed to the constructor:

```
  arguments: ['@path.alias_storage',
    '@path.alias_whitelist',
    '@language_manager',
    '@cache.data']
```

```
public function __construct(
  AliasStorageInterface $storage,
  AliasWhitelistInterface $whitelist,
  LanguageManagerInterface $language_manager,
  CacheBackendInterface $cache) {
```

Obtaining Services: Dependency Injection

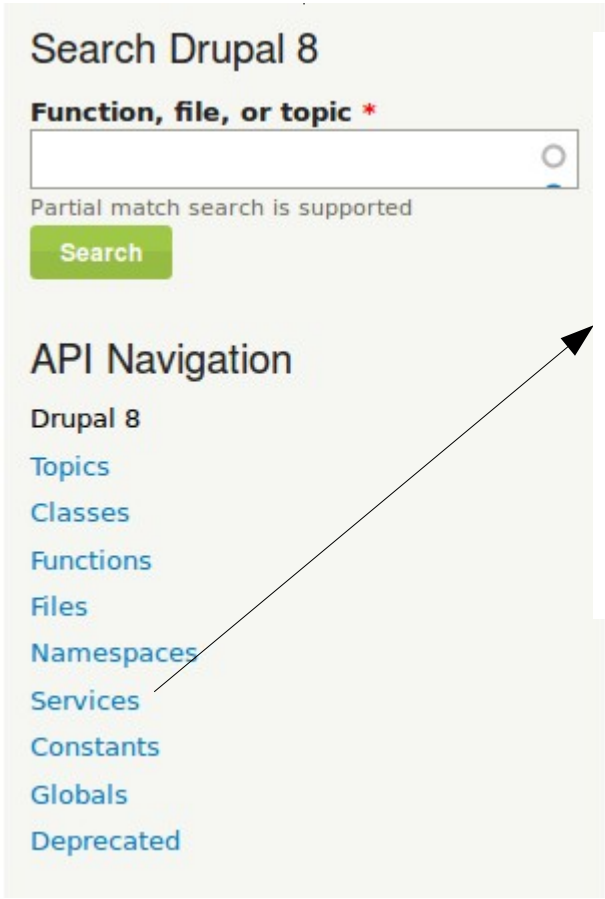
Non-service classes usually have `create()` or `createInstance()` methods called by factory classes, with `$container` arguments. Example from class \Drupal\search\Controller\SearchController:

```
protected $logger;
protected $searchPageRepository;

public static function create(ContainerInterface $container) {
  return new static(
    $container->get('search.search_page_repository'),
    $container->get('logger.factory')->get('search')
  );
}

public function __construct(
  SearchPageRepositoryInterface $search_page_repository,
  LoggerInterface $logger) {
  $this->searchPageRepository = $search_page_repository;
  $this->logger = $logger;
}
```

Discovering Services

Poplar ProductivityWare

api.drupal.org lists services in Drupal Core:



Name contains path

Tag contains

Apply

Contains filters are case sensitive

Name	File	Class
node.admin_path.route_subscriber	core/modules/node/node.services.yml	Drupal\noc \EventSubs \NodeAdm
path.alias_manager	core/core.services.yml	Drupal\Cor \Path\Alias
path.alias_storage	core/core.services.yml	Drupal\Cor \Path\Alias
path.alias_whitelist	core/core.services.yml	Drupal\Cor \Path\Alias

Overriding a Service

1. Locate the default class and interface
2. Define a class that provides the same service in a different way (implement the interface and/or override the class). Call this `\Drupal\foo_bar\MyNewServiceClass`
3. Define a class that implements `\Drupal\Core\DependencyInjection\ServiceModifierInterface`, named `\Drupal\foo_bar\FooBarServiceProvider` (name and namespace are specific!)
4. Put this in the `alter()` method:

```
public function alter(ContainerBuilder $container) {
  $definition =
    $container->getDefinition('name.of.service');
  $definition->setClass(
    'Drupal\foo_bar\MyNewServiceClass');
}
```

Poplar ProductivityWare

Shameless plug