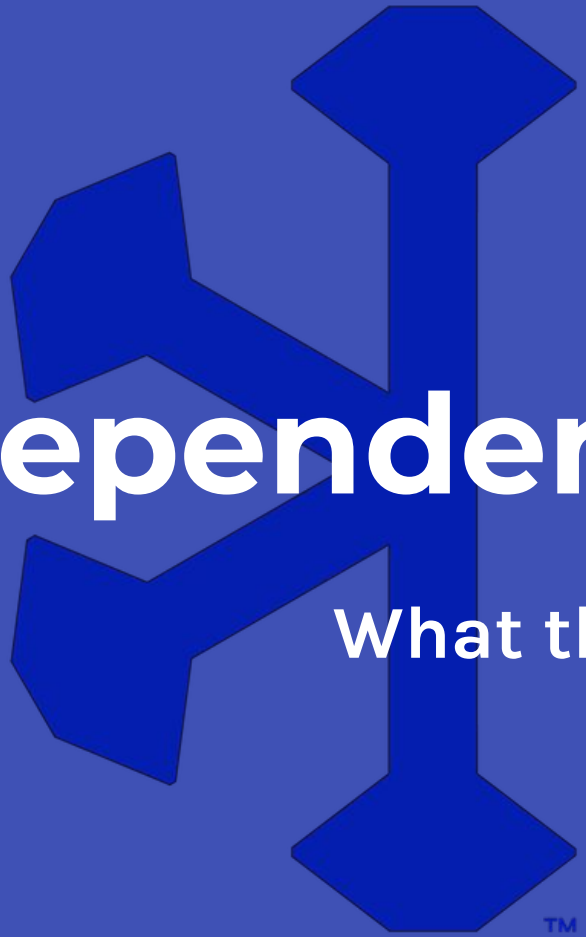


Dependency Injection

What the heck is it?

HELLO!

Salman Jaffery

coldfrontlabs.ca/about#sjaffery

sjpeters79 on drupal.org

Resources

- All code and slides can be found here:

  https://github.com/sjpeters79/drupal-north-2019

- Drupal 8 Service API:

  https://api.drupal.org/api/drupal/services/8.7.x

- Podcasts:
  - Talking Drupal
  - Complete Developer Podcast

Agenda

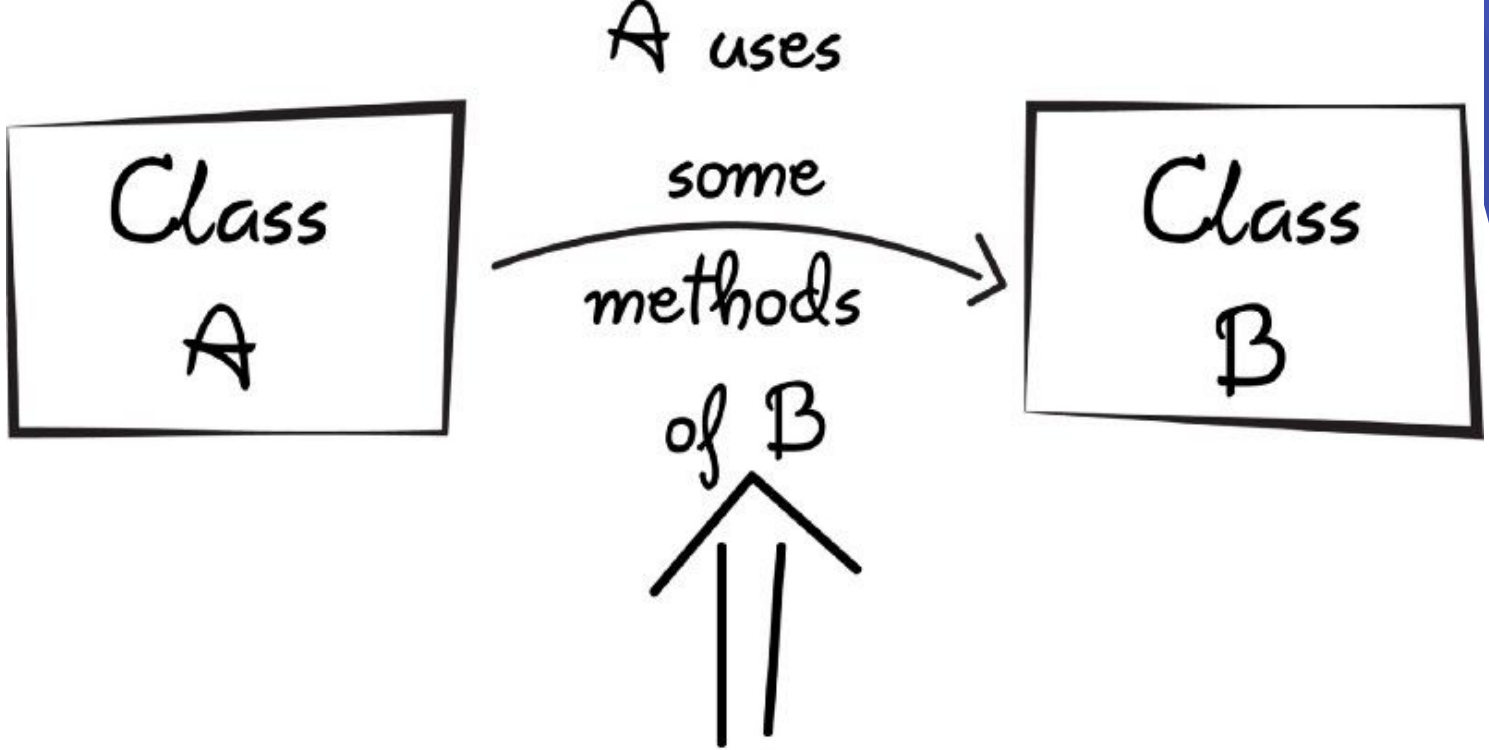
- What is Dependency Injection (DI)?
- Why use DI?
- How do we use DI?
- Symphony
- DI in Drupal 8
- Demo

Coldfront Labs Inc.™

1.

What?

What exactly is dependency injection?



Its a dependency

What is Dependency Injection (DI)?

- Providing requirements to meet an object's dependencies
- Decouple objects so that dependencies are not hard coded

Coldfront Labs Inc.

Purpose of Dependency Injection

- Create the objects
- Know which classes require those objects
- And provide them all those objects

Coldfront Labs Inc.™

2.

Why?

Why do we care about dependency injection?

Why?

- Dependency Injection is how things run in Drupal 8
- Not all code is bootstrapped like in Drupal 7

Advantages

- Leads to self-contained code that’s easy to unit test
- Code requirements self-document
- Allows for more automation
- More organized code base

Disadvantages

- More complex architecture
- Learning curve
- Can be difficult to debug

Coldfront Labs Inc.™

3.

How?

How do we use dependency injection?

Classes are Services

- To organize our code base, we break up our code that provides specific functionality into separate classes.
- These classes are called “Services” as they provide a service for a specific action.
- Services are reusable, easily unit testable, and self documenting by their nature.

Example Service

Class: Drupal\Core\Database\Connection

Factory:

- Drupal\Core\Database::getConnection

Method:

- insert()
- select()
- delete()
- update()

What if our service has dependencies?

How do we ensure that we get them?

For Example, let's assume we have a service that needs database access.

We could define a new one . . .

- If we define a new database connection in each service that runs, there's a lot of duplication.
-
- $conn = \Drupal\Core\Database::getConnection();
- If anything ever changes about the database connection, we'll have to fix it everywhere.
- We're making each service into a standalone application which is a mess.

We could define a global one . . .

```
global $database;
$connection  =
Drupal\Code\Database::getConnection();
db_query(“SELECT * FROM {data}”);
```

- This means anyone using our service needs to know we need the database setup and instantiated before our service runs . . .

Just require it as an argument?

Let's just give our service object a constructor that requires the database connection as an argument.

That way whoever uses our service knows about and has to have one setup before our service runs.

```
protected $connection;
function __constructor(Drupal\Core\Database\Connection $connection) {
  $this->connection = $connection;
}
```

Done!

We've got a service, and it gets its dependencies by forcing the client to inject them into it.

That's dependency injection.

Types of Dependency Injection

- Constructor Injection
- Setter Injection
- Property Injection

Optional Dependencies

For mandatory dependencies, we used constructor dependency injection, but what if we have optional dependencies?

We can use “setter injection”:

```
protected $optional;
function setOptionalDependency($optional) {
  $this->optional = $optional;
}
```

Property Dependencies

Some services can extend other services and take in all their functionality. Whether you are extending or implementing these external services, your service cannot be called unless this exists.

This is known as property dependency:

```
Class MyCustomForm extends FormBase {
...
```

Recap

- Services are some code that does something in a class
- Dependency Injection is explicitly requiring any dependencies
- Service Containers are the metadata about the services and dependencies.
- Service Tags allow you name, categorize and query your services in various ways.

Coldfront Labs Inc.

4.

The Symfony connection

Where does this come from?

Drupal 8 and Symfony

Drupal 8 uses a lot of the Symfony framework

You could say Drupal 8 reflects Drupal features (entities, nodes, fields, blocks) using Symfony's architecture

The concepts of dependency injection, services and service container come from Symfony

Dependency Injection and Symfony

Coldfront Labs Inc.

Services containers are provided by the Dependency Injection Symfony package (symfony/dependency-injection on Packagist)

It defines a library for automatically loading services containers.

```
$container = new ContainerBuilder();
$connection = $container->get('database');
```

Services containers in Symfony

Symfony lets you define your service metadata (containers) in a variety of ways. Because Drupal, we'll only look at YAML.

Service definitions look like this:

```
services:
    service1:
        class: AppBundle\Service\Service1

    service2:
        class: AppBundle\Service\Service2
        arguments: ['@service1']
```

Containers in action

Since the dependency inject container now knows the services and dependencies, it can automatically instantiate dependencies

```
service1:
  class: MyService1

service2:
  class: MyService2
  arguments: [‘@service1’]

service3:
  class: MyService2
  arguments: [‘@service2’]
```

```
$cntr = new ContainerBuilder();
$srv3 = $cntr->get(‘service3’);
```

MyService1, MyService2 and MyService3 are automatically instantiated in the correct order.

Service Tags

Service tags allow you to have multiple services tags with the same value.

```
service1:
  class: MyService1
  tags:
    -{name: service.prime}

service2:
  class: MyService2
  tags:
    -{name: service.prime}
```

These service containers would allow you to have code that ask for an instansiation of all services tagged with “service.prime”.

Each service tag can also include other data.

Recap

- Services are some code that does something in a class
- Dependency Injection is explicitly requiring any dependencies
- Service Containers are the metadata about the services and dependencies.
- Service Tags allow you name, categorize and query your services in various ways.

Coldfront Labs Inc.

5.

How Drupal uses dependency injection

What does this mean for Drupal?

What happened in Drupal 7?

- All modules and code was compiled by PHP during bootstrap
- Shared data was global
- The Drupal run, it would call hook invocations
- Depending on the hook you were implementing, you'd have to know which dependencies were available

Drupal 8

- Uses dependency injection to decide what needs to be instantiated for each page load
- When providing new functionality, instead of implementing a hook, you can provide Drupal core with a tagged service
- When overriding functionality, Drupal core services can be overridden by your own

How does this look?

Service class needs to placed in:

`my_module/src/<namespace>/<classname>.php`

Services containers defined in:

`my_module/my_module.services.yml`

THANKS!

Any questions?