glorpen-di Documentation

Release 1.5.0

Arkadiusz Dzięgiel

Oct 27, 2018

Contents

Another Dependency Injection library for Python.

This package has following three guidelines:

- any class configured by *Container* mechanism should not be modified in any way
- there is no need for external services definition files for *Container*
- no *Container* compiling and service tagging - we have introspection and dynamic language for this task

And so this package provides:

- **no** xml configuration
- **no** annotations (more cons than pros)
- **no** changes to services code

CHAPTER 1

Official repositories

For forking and other funnies.

BitBucket: https://bitbucket.org/glorpen/glorpen-di

GitHub: https://github.com/glorpen/glorpen-di

CHAPTER 2

Supported design patterns

Service instance can be created by:

- factory service
- calling class object with arguments

Instance options can be altered by:

- constructor arguments
- setters
- calling methods
- using configurator service

Each service has defined scope, service cannot request other service from narrower scope.

CHAPTER 3

Contents

3.1 glorpen.di API Documentation

3.1.1 glorpen.di

Dependency injection component.

glorpen.di.**__version__**
Current package version.

class glorpen.di.**Container**
Alias to *glorpen.di.container.Container*.

3.1.2 glorpen.di.container

class glorpen.di.container.**Alias**(*target*)
Alias for service.

class glorpen.di.container.**Container**
Implementation of DIC container.

add_alias(*service*, *alias*)
Adds an alias for given service

add_parameter(*name*, *value*)
Adds a key-value parameter.

add_service(*name*)
Adds service definition to this container.

name argument should be a class, import path, or string if *Service.implementation()* will be used.

Returns: *Service*

get(*svc*)
Gets service instance.

Raises: UnkownServiceException

get_definition(*svc*)
Returns definition for given service name.

get_parameter(*name*)
Gets parameter.

Raises: UnkownParameterException

set_scope_hierarchy(**scopes*)
Sets used scopes hierarchy.

Arguments should be scopes sorted from widest to narrowest. A service in wider scope cannot request service from narrower one.

Default is: [*glorpen.di.scopes.ScopeSingleton*, *glorpen.di.scopes.ScopePrototype*].

Args: classes or instances of *glorpen.di.scopes.ScopeBase*

class glorpen.di.container.**Deffered**(*service=None*, *method=None*, *param=None*)
Class for marking values for lazy resolving.

Values are resolved by *Container* upon service creation.

resolve(*getter*, *param_getter*)
Given (service) getter and param_getter, returns resolved value.

class glorpen.di.container.**Kwargs**(***kwargs*)
Simply wraps given kwargs for later use.

classmethod merge(**args*)
Merges iterable arguments, can be *dict* or *Kwargs* instance. @return: Resulting dict

class glorpen.di.container.**Service**(*name_or_impl*)
Service definition.

When filling arguments for constructor and method calls you can use:

- *my_var__svc=MyClass* - will inject service MyClass to *my_var*
- *my_var__param="my.param"* - will inject parameter named "my.param" to *my_var*

Implementation value for service can be:

- class instance
- string with import path
- callable

call(*method*, *kwargs=None*, ***kwargs_inline*)
Adds a method call after service creation with given arguments.

Returns: *Service*

call_with_signature(*method*, *kwargs=None*, ***kwargs_inline*)
Adds a method call after service creation with given arguments. Arguments detected from function signature are added if not already present.

Returns: *Service*

configurator(*service=None*, *method=None*, *callable=None*, *kwargs=None*, ***kwargs_inline*)
Adds service or callable as configurator of this service.

Args: service + method, callable: given service method/callable will be called with instance of this service.

Returns: *Service*

factory(*service=None*, *method=None*, *callable=None*, *kwargs=None*, ***kwargs_inline*)
Sets factory callable.

Returns: *Service*

implementation(*v*)
Sets service implementation (callable).

Returns: *Service*

kwargs(***kwargs*)
Adds service constructor arguments.

Returns: *Service*

kwargs_from_signature()
Adds arguments found in class signature, based on provided function hints.

Returns: *Service*

kwargs_modifier(*service=None*, *method=None*, *callable=None*, *kwargs=None*, ***kwargs_inline*)
Adds service or callable as constructor arguments modifier of this service.

Args: service + method, callable: given service method/callable will be called with kwargs of this service.

Returns: *Service*

scope(*scope_cls*)
Sets service scope.

Returns: *Service*

set(***kwargs*)
Will `setattr()` given arguments on service.

Returns: *Service*

glorpen.di.container.**fluid**(*f*)
Decorator for applying fluid pattern to class methods and to disallow calling when instance is marked as frozen.

glorpen.di.container.**normalize_name**(*o*)
Gets object "name" for use with *Container*.

Args: o (object): Object to get name for

Returns: str

Raises: Exception

3.1.3 glorpen.di.scopes

Built-in scopes.

class glorpen.di.scopes.**ScopeBase**
Base class for all scopes.

class glorpen.di.scopes.**ScopePrototype**
Scope that creates new instance of given service each time it is requested.

class glorpen.di.scopes.**ScopeSingleton**
Scope that creates instance of given service only once.

3.1.4 glorpen.di.exceptions

Exceptions used by *glorpen.di*.

exception glorpen.di.exceptions.**ContainerException**
Base exception

exception glorpen.di.exceptions.**InjectionException**(*svc_name*, *cls*, *method_name=None*)
Raised when service or its method cannot be created or called

exception glorpen.di.exceptions.**InvalidAliasTargetException**(*name*)
Raised when adding alias to alias or not exisiting service

exception glorpen.di.exceptions.**RecursionException**(*s_def*, *requester_chain*)
Raised when service definition is requiring itself.

exception glorpen.di.exceptions.**ScopeWideningException**(*s_def*, *requester_chain*)
Raised when *Service A* depends on *Service B* from narrower scope.

exception glorpen.di.exceptions.**ServiceAlreadyCreated**(*svc_name*)
Raised when service definition is changed but service is already created by *glorpen.di.container.Container*.

exception glorpen.di.exceptions.**UnknownParameterException**(*name*)
Raised when requesting parameter which is not registered in *glorpen.di.container.Container*.

exception glorpen.di.exceptions.**UnknownScopeException**(*scope*, *svc_name*)
Raised when service has *scope* not assigned to *glorpen.di.container.Container* by *glorpen.di.container.Container.set_scope_hierarchy()*.

exception glorpen.di.exceptions.**UnknownServiceException**(*svc_name*)
Raised when requesting service name which is not registered in *glorpen.di.container.Container*.

3.1.5 glorpen.di.tests

3.2 Usage examples

3.2.1 Injecting services and parameters

```
from glorpen.di import Container

class MyParamService(object):
    pass

class MyService(object):
    def __init__(self, obj, text, value):
        super(MyService, self).__init__()
        print("service instance: %s" % obj)
        print("container parameter: %s" % text)
```

(continues on next page)

(continued from previous page)

```
        print("provided value: %s" % value)

c = Container()

c.add_service(MyParamService)
c.add_parameter('my-param', "value from container")
c.add_service(MyService).kwargs(obj__svc=MyParamService, text__param="my-param",␣
↪value="defined value")

c.get(MyService)
```

Running snippet will print:

```
service instance: <__main__.MyParamService object at 0x7f2fef6e9828>
container parameter: value from container
provided value: defined value
```

3.2.2 Arguments

In cases when you want to use inject parameter already used by internal methods, eg. *glorpen.di.container.Service.call()*, you can pass args with *glorpen.di.container.Kwargs* helper class.

```
svc.configurator(callable=configurator, kwargs=Kwargs(router__svc=Router), other_
↪param="param")
svc.call("some_method", kwargs=Kwargs(router__svc=Router), other_param="param")
```

Arguments defined by *glorpen.di.container.Kwargs* are overriding ones provided by ***kwargs* notation.

3.2.3 Configurators

Configurators are services or callables used to configure main service. Provided callables are given main service instance as first argument.

```
def configurator(obj, some_service):
   obj.some_thing = some_service()
svc.configurator(callable=configurator, some_service__svc=MyClass)
svc.configurator(service=ConfiguratorClass, method="some_method", some_service__
↪svc=MyClass)
```

3.2.4 Factories

Services that create other objects. It is possible to provide parameters/other services from Container to given callables.

```
def factory(some_service):
   return some_service("arg1")
svc1.factory(callable=factory, some_service__svc=MyClass)

class FactoryClass(object):
   def create_new(self, some_service):
   return some_service("arg1")
svc2.factory(service=FactoryClass, method="create_new", some_service__svc=MyClass)
```

3.2.5 Calling methods and settings properties

To call method on service creation:

```
svc.call("some_method", some_service__svc=MyClass)
```

To set properties on service creation:

```
svc.set(my_prop__svc=MyClass)
```

3.3 Using type hints for auto injection

Sometimes it is easier to just auto-fill function arguments, when using Python3 it can be done by arguments type hinting (see `typing` for more information).

You can enable function hints lookup by using *glorpen.di.container.Service.kwargs_from_signature()* for constructor arguments and *glorpen.di.container.Service.call_with_signature()* for methods.

```
from glorpen.di import Container

class MyParamService(object):
    pass

class MyService(object):
    def __init__(self, param:MyParamService):
        super(MyService, self).__init__()
        print("MyService.__init__: %s" % param.__class__.__name__)

    def some_method(self, param:MyParamService):
        print("MyService.some_method: %s" % param.__class__.__name__)

c = Container()

c.add_service(MyParamService)
c.add_service(MyService).kwargs_from_signature().call_with_signature("some_method")

print("Continer.get: %s" % c.get(MyService).__class__.__name__)
```

Snippet will create following output:

```
MyService.__init__: MyParamService
MyService.some_method: MyParamService
Continer.get: MyService
```

3.4 Adding custom scope

You can define new scope by extending *glorpen.di.scopes.ScopeBase* and using *glorpen.di.container.Container.set_scope_hierarchy()*.

```
from glorpen.di.scopes import ScopePrototype, ScopeSingleton, ScopeBase
from random import randint
```

(continues on next page)

(continued from previous page)

```
class RandomScope(ScopeBase):
    """Returns new or cached instances based on random factor."""
    def __init__(self, randomity=3):
        super(RandomScope, self).__init__()
        self.rnd = randomity
        self.instances = {}

    def get(self, creator, name):
        if not name in self.instances or randint(0, self.rnd) == 0:
            self.instances[name] = creator()
        return self.instances[name]

c = Container()

# add scope with parameter
c.set_scope_hierarchy(ScopeSingleton, RandomScope(5), ScopePrototype)

# configure "str" service so we can see instances count
counter = 0
def configurator(kwargs):
    global counter
    kwargs.setdefault("object", "instance number: %d" % counter)
    counter+=1

c.add_service('arg.test').implementation(str)\
    .configurator(args_callable=configurator)\
    .scope(RandomScope)

for i in range(0,10):
    print(c.get("arg.test"))
```

Running script will print:

```
instance number: 0
instance number: 0
instance number: 0
instance number: 0
instance number: 1
instance number: 2
instance number: 2
instance number: 3
instance number: 4
instance number: 4
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index