injectify

Release 0.2.0

Jun 16, 2020

Contents

Welcome to Injectify's documentation! Get started with *Installation* and then get an overview with the *Quickstart*. More information on a specific function, class, or method can be found in the *injectify package* section.

CHAPTER 1

User Guide

This part of the documentation, which is mostly prose, begins with some background information about Flask, then focuses on step-by-step instructions for getting the most out of Injectify.

1.1 Installation

1.1.1 Python Version

Injectify is compatible with Python 3.5 and newer.

1.1.2 Dependencies

These packages will be installed automatically with Injectify. If you don't have `pipenv`, head over the Pipenv website for installation instructions.

- astunparse is an AST unparser for Python.

1.1.3 Pipenv

The recommended way to install the `injectify` package is to simply use pipenv.

```
$ pipenv install injectify
```

Pipenv is a tool that automatically creates and manages a virtual environment for your project.

1.1.4 Virtual Environment

If you prefer, `pip` and `virtualenv` can be used separately. Python comes bundled with the `venv` module to create virtual environments, which you can use.

Create an Environment

Create a project folder and a `venv` folder within:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
$ . venv/bin/activate
```

On Windows:

```
> venv\Scripts\activate
```

Your shell prompt will change to show the name of the activated environment.

Install Injectify

Within the activated environment, use the following command to install Injectify:

```
$ pip install Injectify
```

Injectify is now installed. Check out the *Quickstart* or go to the *Documentation Overview*.

1.2 Quickstart

Eager to get started? This page gives a good introduction to Injectify. Follow *Installation* to set up a project and install Injectify first.

1.2.1 A Minimal Application

Injecting code with Injectify is very simple.

```
from injectify import inject, HeadInjector

def foo(x):
    return x

print(foo(10))  # 10

@inject(target=foo, injector=HeadInjector())
def handler():
    x = 9000

print(foo(10))  # 9000
```

So what does this code do?

1. We begin by importing from the `injectify` module.

2. Next we defined a function `foo` that we will inject code into.
3. When we print the result of `foo(10)`, we get 10.
4. We then use the `inject()` decorator. The first argument is the target object. This is the object that will have code injected into. The second argument is the injector. An injector is used to indicate the point at which the target object should be injected. Here we use the `HeadInjector` for our injector. This injector indicates that the code should be injected at the top of the target object.
5. Then we defined a function `handler`. The body of this function is the code that will be injected into `foo`.
6. Now when we print the result of `foo(10)`, we get 9000.

Thus, after we inject the body of `handler`, the function `foo` then has the following code:

```
def foo(x):
    x = 9000
    return x
```

1.2.2 Injection Points

The first thing we need to be able to do is identify parts of the target object. Let's decorate a function with markers which show some of the areas we are able to easily identify.

```
def bar():
    # HEAD

    def wrapper():  # NESTED
        pass

    x = 10  # FIELD

    if x > 10:
        # RETURN
        return True
    else:
        # RETURN
        return False

    # TAIL
```

The markers indicate parts of the object's anatomy:

- **HEAD** indicates the point at the top of the target object.
- **TAIL** indicates the point at the bottom of the target object.
- **FIELD** indicates the point at a field's assignment.
- **RETURN** indicates the point before a return statement.
- **NESTED** indicates a nested function.

Note: Not all Python objects have all these injection points. For example, a module does not have a return statement, so a module has no RETURN marker.

1.2.3 Injectors

The injector you use tells Injectify the injection point to use when merging the code inside the target object. Each of the markers above has a corresponding injector.

For more information on injectors you can visit the *injectify.injectors module* documentation.

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 injectify package

2.1.1 Submodules

2.1.2 injectify.api module

2.1.3 injectify.injectors module

2.1.4 injectify.structures module

2.1.5 injectify.utils module

2.1.6 Module contents

CHAPTER 3

Project Info

Design notes, legal information and changelog are here for the interested.

3.1 History

3.1.1 0.2.0

- Add more tests
- Add documentation
- Bug fixes

3.1.2 0.1.1

- Fix dill dependency version

3.1.3 0.1.0

- Conception!

3.2 License

BSD 3-Clause License

Copyright (c) 2020, Mitchell Marsden All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

CHAPTER 4

Index

- genindex
- modindex