Руководство по работе с PyPI

Выпуск

ROBIN RPA Team

дек. 24, 2020

Содержание:


ROBIN
ROBOTIC INTELLIGENCE

Глава 1

Как работать с PYPI

1.1 Оглавление

1.1.1 Сборка пакета (в wheel)

Для корректной сборки пакета нужно создать в корне проекта файл `setup.py` со следующим содержимым:

```
from setuptools import setup

setup(
    name='vision', # Название пакета. Для установки пакета введите команду в консоли: pip␣
↪install vision (по умолчанию ставится последняя версия пакета)
    version='1.0', # Версия пакета (в данном случае 1.0). Для установки конкретной версии␣
↪введите команду в консоли: pip install vision==1.0
    packages=['vision', 'vision.actions'], # Модули, которые войдут в пакет (необязательные␣
↪можно при желании исключить)
    python_requires='~=3.6',  # Требуемая версия Python - в данном случае 3.6 (но не 4, так␣
↪как тильда не позволяет инкрементировать версии)
    install_requires=[
        'Pillow==6.2.2',
        'numpy',
        'pytesseract'
    ],  # Требуемые зависимости пакета для его корректной работы
```

```
    url='', # URL адрес проекта, например, на github (не обязательно)
    license='', # Лицензия (не обязательно)
    author='ROBIN Platform', # Автор пакета (используется в метаданных pip, не обязательно)
    author_email='', # Почтовый адрес автора пакета (в случае с официальным PyPI репозиторием -
↪ могут приходить уведомления о том, что какие-то модули устарели / содержат уязвимости, в␣
↪остальных случаях используется просто для обратной связи)
    description='Actions for working with OCR and computer vision' # Описание пакета (не␣
↪обязательно, используется в метаданных pip)
)
```

После создания `setup.py`, устанавливаем и обновляем все необходимые зависимости для корректной сборки пакета.

```
pip install --upgrade wheel setuptools
```

Для сборки готового пакета (wheel) запускаем следующую команду:

```
python setup.py sdist bdist_wheel
```

В результате работы этой команды должны создаться две директории - `build` и `dist`. Внутри dist будет .whl файл нашего собранного пакета.

1.1.2 Загрузка пакета в сторонний PyPI-репозиторий

Для загрузки пакета в репозиторий нам потребуется установить `twine`. Выполняем следующую команду:

```
pip install twine
```

После установки twine, загружаем наш новый пакет в репозиторий следующей командой (адрес репозитория может различаться):

```
twine upload --repository-url http://172.28.1.250:8083/repository/pypi-releases/simple/ dist/*
```

Twine запросит логин и пароль от репозитория. Вводим их и пакет успешно загружается в репозиторий.

Чтобы Twine не запрашивал логин и пароль в интерактивном режиме, можно загрузить пакет в репозиторий следующей командой:

```
twine upload -u [login] -p [password] --repository-url http://172.28.1.250:8083/repository/
↪pypi-releases/simple/ dist/*
```

где [login] и [password] - логин и пароль от репозитория.

1.1.3 Установка пакета из стороннего репозитория

Чтобы установить пакет (в нашем случае `vision`) из стороннего репозитория, используем следующую команду

```
pip install -i http://[login]:[password]@172.28.1.250:8083/repository/pypi-releases/simple/ --
↪trusted-host 172.28.1.250 vision
```

где [login] и [password] - ваши логин и пароль от репозитория.

1.1.4 Использование установленного пакета

Вариант 1. Импорт целого пакета

```
import vision

image_url = "test.jpg"
print(vision.actions.read_text_from_image(image, lang='rus'))
```

Вариант 2. Импорт отдельных методов/модулей пакета.

```
from vision.actions import read_text_from_image

image_url = "test.jpg"
print(read_text_from_image(image, lang='rus'))
```

1.1.5 Общая информация о пакетах

Структура правильного PyPI-пакета должна быть следующей (упрощена для наглядности).

```
|   .gitignore
|   .gitlab-ci.yml
|
```

+—ExistsOnScreen | | setup.py | | __init__.py | | | —ExistsOnScreen | action.py | Robin.Vision.ExistsOnScreen-Python.robin-impinfo | __init__.py | +—FindOnScreen | | setup.py | | __init__.py | | | —FindOnScreen | action.py | Robin.Vision.FindOnScreen-Py.robin-impinfo | __init__.py | +—ReadTextFromImage | | setup.py | | __init__.py | | | —ReadTextFromImage | action.py | Robin.Vision.ReadTextFromImage-Py.robin-impinfo | __init__.py | +—RecognizeByTemplate | | setup.py | | __init__.py | | | —RecognizeByTemplate | action.py | Robin-Py.Vision.RecognizeByTemplate-Py.robin-impinfo | __init__.py | +—Utils | | setup.py | | __init__.py | | | —Utils | utils.py | __init__.py | +—WaitForAction | | setup.py | | __init__.py | | | —WaitForAction | action.py | Robin-Py.Vision.WaitForAction-Py.robin-impinfo | __init__.py | +—WaitForDisappear | | setup.py | | __init__.py | | | —WaitForDisappear | action.py | Robin-Py.Vision.WaitForDisappear-Py.robin-impinfo | __init__.py |

В директории .egg-info хранится мета-информация об установленном пакете.

```
├── vision.egg-info
    ├── PKG-INFO  # Информация о пакете
    ├── SOURCES.txt  # Информация о содержимом пакета - какие модули в него входят
    ├── dependency_links.txt  # Информация о зависимостях этого пакета (если они присутствуют)
    └── top_level.txt  # Название пакета
```

Глава 2

Восстановление пакетов

Пакетный менеджер pip самостоятельно определяет и подгружает зависимости пакета (их мы указали в списке install_requires в листинге setup.py выше).

Для установки пакета со всеми его зависимостями достаточно ввести следующую команду:

```
pip install -i http://[login]:[password]@172.28.1.250:8083/repository/pypi-releases/simple/ --
↪trusted-host 172.28.1.250 vision
```

Установка пакета из локального файла (wheel) `.whl` ничем не отличается от обычной установки пакета, разве что при установке wheel нужно установить пакет `wheel`

```
pip install wheel
```

и в команде `pip install` указать путь к этому файлу

```
pip install ./dist/vision-1.0-py3-none-any.whl
```

Так, если в wheel'е уже на этапе заполнения файла setup.py были прописаны зависимости в install_requires, то pip подгрузит их для wheel'а автоматически.

Допустим, у нас есть wheel'ы в локальной директории `/home/thealchemist/wheels/`, и один из них - vision. Чтобы установить конкретный пакет - vision - мы выполняем следующую команду

```
pip install --no-index --find-links /home/thealchemist/wheels/ vision
```

Ключ `--no-index` указывает pip не обращаться к Python Global PyPI, а устанавливать исключительно из локальной среды.

Чтобы установить все wheel'ы из директории, выполняем команду

```
pip install --no-index --find-links /home/thealchemist/wheels/ *.whl
```

Допустим, у нас есть следующая структура:

```
├── actions
│   ├── __init__.py
│   ├── exists_on_screen.py
│   ├── find_on_screen.py
│   ├── read_text_from_image.py
│   ├── recognize_by_template.py
```

```
    ├── wait_for_action.py
    └── wait_for_disappear.py
```

Предположим, мы хотим иметь возможность импортировать наши экшны следующим образом (просто и логически верно):

```
from actions import read_text_from_image

print(read_text_from_image('awesome_image.jpg'))
```

Дело в том, что любой `.py` файл в Python является модулем.
Так как у нас уже присутствует файл `read_text_from_image.py`,
то вместо конкретной функции у нас будет импортирован целый модуль,

соответственно при вызове модуля будет выброшена ошибка (модуль нельзя вызвать, это не метод).

Чтобы избежать таких неприятностей, мы в __init__.py прописываем следующие импорты:

```
# Переносы сделаны для наглядности объяснения
from  # Из
    .read_text_from_image  # модуля read_text_from_image в текущей директории (точка перед␣
↪названием модуля)
import  # мы импортируем
    read_text_from_image  # метод/функцию/объект read_text_from_image

# остальные импорты
from .exist_on_screen import exists_on_screen
from .recognize_by_template import recognize_by_template
# и так далее
```

Важно отметить, что из модуля мы импортируем **метод** с *таким же* названием.

Глава 3

Поставка Python для инсталлятора [Windows]

Все исполняемые среды Python любой версии можно загрузить в локальный репозиторий с https://www.python.org/ftp/python/

Добавляем Python в инсталлятор следующим образом: Скачиваем установочный exe с питоньей средой исполнения с глобального / нашего локального репозитория, и запаковываем его в Inno Setup (где в процессе установки всего потом вызываем). Для тихой установки Python вызываем инсталлятор с флагом /quiet и дополнительными параметрамии при необходимости. Список всех дополнительных параметров можно найти на https://docs.python.org/3.6/using/windows.html#installing-without-ui

Для того, чтобы свежеустановленный Python заработал, надо создать для него локальную переменную среды, в которую прописать путь к `%PYTHON_DIR%/bin/`, где `%PYTHON_DIR%` - место, куда распакован архив с Python.

Дальше можно вызывать наш конкретный свежеустановленный Python следующей командой (предположим, что переменная среды называется `PYLOCALENV`)

```
%PYLOCALENV%/python --version
```

Соответственно все команды вроде `pip`, `easy_install`, `twine` вызываем с контекстом %PYLOCALENV%

```
%PYLOCALENV%/pip install requests
```

Все установленные пакеты в локальный Python можно найти по пути %PYTHON_DIR%/site-packages/ (в случае портативной установки) или %APPDATA%/Roaming/Python/Python38/site-packages/ (в случае установки с инсталлятора)

Глава 4

Генерация protobuf классов

Разберем создание протобафного класса. В качестве примера клонируем репозиторий:

```
git clone http://git-server/contracts/protocols.git -b master
```

Все команды выполняются в cmd в Windows. В каждой директории с proto файлами создаём папку python, в которой будут хранится сгенерированные классы:

```
cd protocols

cd proto\Base
mkdir python
protoc -I=. -I=.\.. -I=.\..\..\include\ --python_out=python *.proto

cd ..\ExecutorAgent
mkdir python
protoc -I=. -I=.\.. -I=.\..\..\include\ --python_out=python *.proto

cd ..\AgentOrchestrator
mkdir python
protoc -I=. -I=.\.. -I=.\..\..\include\ --python_out=python *.proto

cd ..\OrchestratorUI
mkdir python
protoc -I=. -I=.\.. -I=.\..\..\include\ --python_out=python *.proto

cd ..\StudioAgent
mkdir python
protoc -I=. -I=.\.. -I=.\..\..\include\ --python_out=python *.proto
```

В `.gitlab-ci.yml` команды будут аналогичными.

Пример кода для генерации классов в `.gitlab-ci.yml`:

```
cd proto\Base
mkdir python
protoc -I=./ -I=./../ -I=./../../include/ *.proto --python_out=./python/
```