Использование и вызов

10.1 Вызов `pytest` с помощью `python -m pytest`

Тестирование можно запустить из командной строки интерпретатора `Python`, используя команду:

```
python -m pytest [...]
```

В отличие от запуска напрямую командой `pytest [...]`, запуск через `Python` добавит текущий каталог в `sys.path`.

.2 Статусы завершения

Выполнение `pytest` может генерировать один из следующих статусов завершения:

Exit code 0 Все тесты были собраны и успешно прошли

Exit code 1 Тесты были собраны и запущены, но некоторые из них упали

Exit code 2 Выполнение тестов было прервано пользователем

Exit code 3 Во время выполнения тестов произошла внутренняя ошибка

Exit code 4 Ошибка запуска `pytest` из командной строки

Exit code 5 Не удалось собрать тесты (тесты не найдены)

Коллекция статусов представлена перечислением _pytest.config.ExitCode. Статусы завершения

являются частью публичного API, их можно импортировать и использовать непосредственно:

```
from pytest import ExitCode
```

Примечание: Для настройки кода завершения сценария, особенно когда тесты не удалось собрать, можно использовать плагин pytest-custom_exit_code.

10.3 Получение помощи по версии, параметрам, переменным окружения

```
pytest --version   # показывает версию и место, откуда импортирован ``pytest``
pytest --fixtures  # показывает доступные встроенные функции
pytest -h | --help # показывает помощь по командной строке и параметры конфигруационного файла
```

10.4 Остановка после первых N падений

Чтобы остановить процесс тестирования после первых N падений, используются параметры:

```
pytest -x           # остановка после первого упавшего теста
pytest --maxfail=2  # остановка после первых двух упавших тестов
```

10.5 Выбор выполняемых тестов

`pytest` поддерживает несколько способов выбора и запуска тестов из командной строки.

Запуск тестов модуля

```
pytest test_mod.py
```

Запуск тестов из директории

```
pytest testing/
```

Запуск тестов, удовлетворяющих ключевому выражению

```
pytest -k "MyClass and not method"
```

Эта команда запустит тесты, имена которых удовлетворяют заданному строковому выражению (без учета регистра). Строковые выражения могут включать операторы `Python`, которые используют имена файлов, классов и функций в качестве переменных. В приведенном выше примере будет запущен тест `MyClass.test_something`, но не будет запущен тест `TestMyClass.test_method_simple`.

Запуск тестов по идентификаторам узлов

Каждому собранному тесту присваивается уникальный идентификатор `nodeid`, который состоит из имени файла модуля, за которым следуют спецификаторы, такие как имена классов, имена функций и параметры из параметризации, разделенные символами `::`:

Чтобы запустить конкретный тест из модуля, выполните:

```
pytest test_mod.py::test_func
```

Еще один пример спецификации тестового метода в командной строке:

```
pytest test_mod.py::TestClass::test_method
```

Запуск маркированных тестов

```
pytest -m slow
```

Будут запущены тесты, помеченные декоратором `@pytest.mark.slow`.

Подробнее см. *marks*.

Запуск тестов из пакетов

```
pytest --pyargs pkg.testing
```

Будет импортирован пакет `pkg.testing`, и его расположение в файловой системе будет использовано для поиска и запуска тестов.

10.6 Изменение вывода сообщений трассировки

Примеры вывода:

```
pytest --showlocals # показывать локальные переменные в сообщениях
pytest -l           # показывать локальные переменные в сообщениях (краткий вариант)
pytest --tb=auto    # (по умолчанию) "расширенный" вывод для первого и
                    # последнего сообщений, и "короткий" для остальных
pytest --tb=long    # исчерпывающий, подробный формат сообщений
pytest --tb=short   # сокращенный формат сообщений
pytest --tb=line    # только одна строка на падение
pytest --tb=native  # стандартный формат библиотеки Python
pytest --tb=no      # никаких сообщений
```

Использование `--full-trace` приводит к тому, что при ошибке печатаются очень длинные трассировки (длиннее, чем при `--tb=long`). Параметр также гарантирует, что сообщения трассировки будут напечатаны при **прерывании выполнения с клавиатуры** с помощью Ctrl+C. Это очень полезно, если тесты занимают слишком много времени, и вы прерываете их с клавиатуры с помощью Ctrl+C, чтобы узнать, где они зависли. По умолчанию при прерывании вывод не будет показан (поскольку исключение `KeyboardInterrupt` будет поймано `pytest`). Используя этот параметр, вы можете быть уверены, что увидите трассировку.

2.7 Детализация сводного отчета

Флаг `-r` можно использовать для отображения «краткой сводной информации по тестированию» в конце тестового сеанса, что упрощает получение четкой картины всех сбоев, пропусков, xfails и т. д.

По умолчанию для списка сбоев и ошибок используется добавочная комбинация `fE`.

Пример:

```
# content of test_example.py
import pytest


@pytest.fixture
def error_fixture():
    assert 0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
def test_ok():
    print("ok")


def test_fail():
    assert 0


def test_error(error_fixture):
    pass


def test_skip():
    pytest.skip("skipping this test")


def test_xfail():
    pytest.xfail("xfailing this test")


@pytest.mark.xfail(reason="always xfail")
def test_xpass():
    pass
```

```
$ pytest -ra
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 6 items

test_example.py .FEsxX                                                 [100%]

================================== ERRORS ==================================
_____________________ ERROR at setup of test_error _____________________

    @pytest.fixture
    def error_fixture():
>       assert 0
E       assert 0

test_example.py:6: AssertionError
================================= FAILURES =================================
_______________________________ test_fail _______________________________

    def test_fail():
>       assert 0
E       assert 0

test_example.py:14: AssertionError
======================== short test summary info ========================
SKIPPED [1] $REGENDOC_TMPDIR/test_example.py:22: skipping this test
XFAIL test_example.py::test_xfail
  reason: xfailing this test
XPASS test_example.py::test_xpass always xfail
ERROR test_example.py::test_error - assert 0
```

(continues on next page)

(продолжение с предыдущей страницы)

```
FAILED test_example.py::test_fail - assert 0
== 1 failed, 1 passed, 1 skipped, 1 xfailed, 1 xpassed, 1 error in 0.12s ===
```

Параметр `-r` принимает ряд символов после себя. Использованный выше символ `a` означает "все, кроме успешных».

Вот полный список доступных символов, которые можно использовать:

- `f` - упавшие (добавляет раздел FAILED)
- `E` - ошибки (добавляет раздел ERROR)
- `s` - пропущенные (добавляет раздел SKIPPED)
- `x` - тесты XFAIL (добавляет раздел XFAIL)
- `X` - тесты XPASS (добавляет раздел XPASS)
- `p` - успешные (passed)
- `P` - успешные (passed) с выводом

Есть и специальные символы для пропуска отдельных групп:

- `a` - выводить все, кроме `pP`
- `A` - выводить все
- `N` - ничего не выводить (может быть полезным, поскольку по умолчанию используется комбинация `fE`).

Можно использовать более одного символа. Например, для того, чтобы увидеть только упавшие и пропущенные тесты, можно выполнить:

```
$ pytest -rfs
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 6 items

test_example.py .FEsxX                                                [100%]

================================== ERRORS ==================================
_____________________ ERROR at setup of test_error _____________________

    @pytest.fixture
    def error_fixture():
>       assert 0
E       assert 0

test_example.py:6: AssertionError
================================= FAILURES =================================
_______________________________ test_fail ________________________________

    def test_fail():
>       assert 0
E       assert 0

test_example.py:14: AssertionError
======================== short test summary info ========================
```

(continues on next page)

(продолжение с предыдущей страницы)

```
FAILED test_example.py::test_fail - assert 0
SKIPPED [1] $REGENDOC_TMPDIR/test_example.py:22: skipping this test
== 1 failed, 1 passed, 1 skipped, 1 xfailed, 1 xpassed, 1 error in 0.12s ===
```

Использование `p` добавляет в сводный отчет успешные тесты, а `P` добавляет дополнительный раздел «пройдены" (PASSED) для тестов, которые прошли, но перехватили вывод:

```
$ pytest -rpP
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-5.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 6 items

test_example.py .FEsxX                                                 [100%]

================================== ERRORS ==================================
_____________________ ERROR at setup of test_error _____________________

    @pytest.fixture
    def error_fixture():
>       assert 0
E       assert 0

test_example.py:6: AssertionError
================================= FAILURES =================================
_______________________________ test_fail ________________________________

    def test_fail():
>       assert 0
E       assert 0

test_example.py:14: AssertionError
================================= PASSES ===================================
_______________________________ test_ok ____________________________________
--------------------------- Captured stdout call ---------------------------
ok
======================== short test summary info ========================
PASSED test_example.py::test_ok
== 1 failed, 1 passed, 1 skipped, 1 xfailed, 1 xpassed, 1 error in 0.12s ===
```

2.8 Запуск отладчика PDB (Python Debugger) при падении тестов

`python` содержит встроенный отладчик PDB (Python Debugger). `pytest` позволяет запустить отладчик с помощью параметра командной строки:

```
pytest --pdb
```

Использование параметра позволяет запускать отладчик при каждом падении теста (или прерывании его с клавиатуры). Часто хочется сделать это для первого же упавшего теста, чтобы понять причину его падения: