Публикации

Новости

Пользователи

Хабы

Компании

Песочница





Регис



🧸 AlekSandrDr вчера в 17:14

# Python Testing с pytest. Использование pytest с другими инструмента ГЛАВА 7

Автор оригинала: Okken Brian

Tutorial

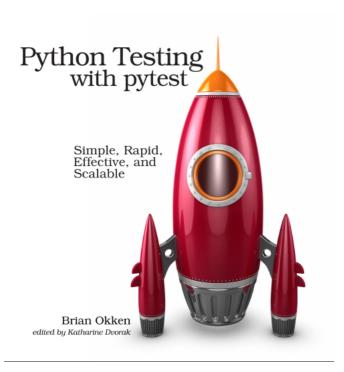
Python

Перевод

🧲 Вернуться

Обычно pytest используется не самостоятельно, а в среде тестирования с другими инструментами. В этой главе рассматриваются с инструменты, которые часто используются в сочетании с pytest для эффективного и результативного тестирования. Хотя это от не исчерпывающий список, обсуждаемые здесь инструменты дадут вам представление о вкусе силы смешивания pytest с другими инструментами.





Примеры в этой книге написаны с использованием Python 3.6 и pytest 3.2. pytest 3.2 поддерживает Python 2.6, 2.7 и Python 3.3+.

Исходный код для проекта Tasks, а также для всех тестов, показанных в этой книге, доступен по ссылке на веб-странице книги в pragprog.com. Вам не нужно загружать исходный код, чтобы понять тестовый код; тестовый код представлен в удобной форме в примера Но что бы следовать вместе с задачами проекта, или адаптировать примеры тестирования для проверки своего собственного проекта (г у вас развязаны!), вы должны перейти на веб-страницу книги и скачать работу. Там же, на веб-странице книги есть ссылка для сообщени errata и дискуссионный форум.

Под спойлером приведен список статей этой серии.

Оглавление

## pdb: Debugging Test Failures

Модуль pdb является отладчиком Python в стандартной библиотеке. Вы используете --pdb, чтобы pytest начал сеанс отладки в точке сбоя. Давайте посмотрим на pdb в действии в контексте проекта Tasks.

В "Параметризации Фикстур" на странице 64 мы оставили проект Tasks с несколькими ошибками:

Прежде чем мы рассмотрим, как pdb может помочь нам отладить этот тест, давайте взглянем на доступные параметры pytest, чтобы ускори отладку ошибок теста, которые мы впервые рассмотрели в разделе "Использование Опций" на стр.9:

- --tb=[auto/long/short/line/native/no]: Управляет стилем трассировки.
- -v / --verbose: Отображает все имена тестов, пройденных или не пройденных.
- -1 / --showlocals: Отображает локальные переменные рядом с трассировкой стека.
- -lf / --last-failed: Запускает только тесты, которые завершились неудачей.
- -x / --exitfirst: Останавливает тестовую сессию при первом сбое.
- --рdb: Запускает интерактивный сеанс отладки в точке сбоя.

#### Installing MongoDB

Как упомянуто в главе 3, "Pytest Fixtures", на странице 49, для запуска тестов MongoDB требуется установка моngoDB и румоngo.

Я тестировал версию Community Server, найденную по адресу https://www.mongodb.com/download-center. pymongo устанавливается с pip:p: install pymongo. Однако это последний пример в книге, где используется MongoDB. Чтобы опробовать отладчик без использования Mongo можно выполнить команды pytest из code/ch2/, так как этот каталог также содержит несколько неудачных тестов.

Мы просто запустили тесты из code/ch3/c, чтобы убедиться, что некоторые из них не работают. Мы не видели tracebacks или имен тестов, что --tb=no отключает трассировку, и у нас не было включено --verbose. Давайте повторим ошибки (не более трех) с подробным текстом:

Теперь мы знаем, какие тесты провалились. Давайте рассмотрим только один из них, используя -x, включив трассировку, не используя --t показывая локальные переменные с -1:

```
def test add returns valid id(tasks db):
       """tasks.add(<valid task>) should return an integer."""
       # GIVEN an initialized tasks db
       # WHEN a new task is added
       # THEN returned task id is of type int
       new task = Task('do something')
       task id = tasks.add(new task)
> assert isinstance(task id, int)
E AssertionError: assert False
E + where False = isinstance(ObjectId('59783baf8204177f24cb1b68'), int)
new task = Task(summary='do something', owner=None, done=False, id=None)
task_id = ObjectId('59783baf8204177f24cb1b68')
tasks db = None
tests/func/test_add.py:16: AssertionError
!!!!!!!!! Interrupted: stopping after 1 failures !!!!!!!!!!
======= 1 failed, 54 deselected in 2.47 seconds ========
```

Довольно часто этого достаточно, чтобы понять почему случился провал теста. В этом конкретном случае довольно ясно, что task\_id не яг целым числом—это экземпляр ObjectId. ObjectId — это тип, используемый MongoDB для идентификаторов объектов в базе данных. Мое намерение со слоем tasksdb\_pymongo.py было скрыть определенные детали реализации MongoDB от остальной части системы. Понятно, этом случае это не сработало.

Тем не менее, мы хотим посмотреть, как использовать pdb с pytest, так что давайте представим, что неясно, почему этот тест не удался. Ми можем сделать так, чтобы pytest запустил сеанс отладки и запустил нас прямо в точке сбоя с помощью --pdb:

```
$ pytest -v --lf -x --pdb
        ======== test session starts ============
run-last-failure: rerun last 42 failures
collected 96 items
tests/func/test add.py::test add returns valid id[mongo] FAILED
tasks db = None
   def test add returns valid id(tasks db):
      """tasks.add(<valid task>) should return an integer."""
      # GIVEN an initialized tasks db
      # WHEN a new task is added
       # THEN returned task_id is of type int
      new task = Task('do something')
      task id = tasks.add(new task)
> assert isinstance(task id, int)
E AssertionError: assert False
E + where False = isinstance(ObjectId('59783bf48204177f2a786893'), int)
tests/func/test add.py:16: AssertionError
> /path/to/code/ch3/c/tasks_proj/tests/func/test_add.py(16)
> test add returns valid id()
-> assert isinstance(task id, int)
(Pdb)
```

Теперь, когда мы находимся в приглашении (Pdb), у нас есть доступ ко всем интерактивным функциям отладки pdb. При просмотре сбоев у регулярно использую эти команды:

- p/print expr: Печатает значение exp.
- pp expr: Pretty печатает значение expr.
- 1/list: Перечисляет точку сбоя и пять строк кода выше и ниже.
- 1/list begin, end: Перечисляет конкретные номера строк.
- a/args: Печатает аргументы текущей функции с их значениями.
- u/up: Перемещается на один уровень вверх по трассе стека.
- d/down: Перемещается вниз на один уровень в трассировке стека.

• q/quit: Завершает сеанс отладки.

Другие навигационные команды, такие как step и next, не очень полезны, так как мы сидим прямо в операторе assert. Вы также можете просввести имена переменных и получить значения.

Можно использовать p/print expr аналогично параметру -1/--showlocals для просмотра значений в функции:

```
(Pdb) p new_task
Task(summary='do something', owner=None, done=False, id=None)
(Pdb) p task_id
ObjectId('59783bf48204177f2a786893')
(Pdb)
```

Теперь можно выйти из отладчика и продолжить тестирование.

Если бы мы не использовали -x, pytest бы снова открыл Pdb в следующем теста. Дополнительные сведения об использовании модуля pdb доступны в документации Python.

### Coverage.py: Определение объема тестируемого кода

Покрытие кода является показателем того, какой процент тестируемого кода тестируется набором тестов. Когда вы запускаете тесты для п «Tasks», некоторые функции «Tasks» выполняются с каждым тестом, но не со всеми.

Инструменты покрытия кода отлично подходят для того, чтобы сообщить вам, какие части системы полностью пропущены тестами.

Coverage.py является предпочтительным инструментом покрытия Python, который измеряет покрытие кода.

Вы будете использовать его для проверки кода проекта Tasks с помощью pytest.

Что бы использовать coverage.py нужно его установить. Не помешает установить плагин под названием pytest-cov, который позволит вам вызывать coverage.py от pytest с некоторыми дополнительными опциями pytest. Поскольку coverage является одной из зависимостей pyte достаточно установить pytest-cov и он притянет за собой coverage.py:

```
$ pip install pytest-cov
Collecting pytest-cov
Using cached pytest_cov-2.5.1-py2.py3-none-any.whl
Collecting coverage>=3.7.1 (from pytest-cov)
Using cached coverage-4.4.1-cp36-cp36m-macosx_10_10_x86
...
Installing collected packages: coverage, pytest-cov
Successfully installed coverage-4.4.1 pytest-cov-2.5.1
```

Давайте запустим отчет о покрытии для второй версии задач. Если у вас все еще установлена первая версия проекта Tasks, удалите ее и установите версию 2:

```
$ pip uninstall tasks
Uninstalling tasks-0.1.0:
   /path/to/venv/bin/tasks
   /path/to/venv/lib/python3.6/site-packages/tasks.egg-link
Proceed (y/n)? y
   Successfully uninstalled tasks-0.1.0
$ cd /path/to/code/ch7/tasks_proj_v2
$ pip install -e .
Obtaining file:///path/to/code/ch7/tasks_proj_v2
...
Installing collected packages: tasks
Running setup.py develop for tasks
```

```
Successfully installed tasks

$ pip list
...
tasks (0.1.1, /path/to/code/ch7/tasks_proj_v2/src)
...
```

Теперь, когда установлена следующая версия задач, можно запустить отчет о базовом покрытии:

```
$ cd /path/to/code/ch7/tasks proj v2
$ pytest --cov=src
plugins: mock-1.6.2, cov-2.5.1
collected 62 items
tests/func/test add.py ...
tests/func/test_add_variety.py ......
tests/func/test_add_variety2.py .....
tests/func/test_api_exceptions.py ......
tests/func/test_unique_id.py .
tests/unit/test_cli.py .....
tests/unit/test_task.py ....
----- coverage: platform darwin, python 3.6.2-final-0 ------
                    Stmts Miss Cover
Name
2 0 100%
                               72%
src\tasks\cli.py
                     45 14
                              69%
src\tasks\config.py 18 12 33%
src\tasks\tasksdb_pymongo.py 74 74
                               0%
                     32 4 88%
src\tasks\tasksdb_tinydb.py
TOTAL
                     250 126 50%
```

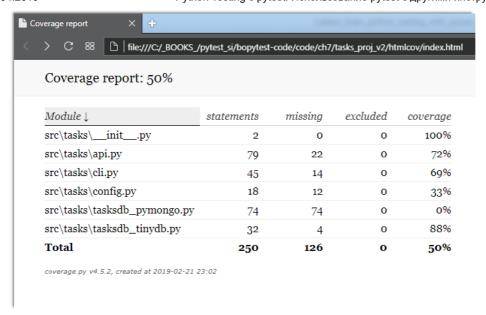
Поскольку текущий каталог является tasks\_proj\_v2, а тестируемый исходный код находится в src, добавление опции --cov=src создает от покрытии только для этого тестируемого каталога.

Как видите, некоторые файлы имеют довольно низкий и даже 0%, охват. Это полезные напоминания: tasksdb\_pymongo.py 0%, потому что м отключили тестирование для MongoDB в этой версии. Некоторые из них довольно низкие. Проект, безусловно, должен будет поставить тес всех этих областей, прежде чем он будет готов к прайм-тайм.

Я полагаю, что несколько файлов имеют более высокий процент покрытия: api.py и tasksdb\_tinydb.py. Давайте посмотрим на tasksdb\_tinydb.py и посмотрим, чего не хватает. Думаю, что лучший способ сделать это — использовать отчеты HTML.

Если вы снова запустите coverage.py с параметром --cov-report=html, будет создан отчет в формате HTML:

Затем можно открыть htmlcov/index.html в браузере, который показывает вывод на следующем экране:



Щелчок по tasksdb\_tinydb.py покажет отчет для одного файла. В верхней части отчета отображается процент покрытые строки, плюс скол строк охвачено и сколько нет, как показано на следующем экране:



Прокручивая вниз, вы можете увидеть пропущенные строки, как показано на следующем экране:

```
def list_tasks(self, owner=None): # type (str) -> list[dict]
            """Return list of tasks."""
33
           if owner is None:
34
               return self._db.all()
            else:
              return self._db.search(tinydb.Query().owner == owner)
36
38
       def count(self): # type () -> int
            """Return number of tasks in db."""
40
            return len(self._db)
41
42
       def update(self, task_id, task): # type (int, dict) -> ()
43
             ""Modify task in db with given task id."
44
           self._db.update(task, eids=[task_id])
46
       def delete(self, task_id): # type (int) -> ()
            """Remove a task from db with given task_id."""
48
           self._db.remove(eids=[task_id])
50
       def delete all(self):
              "Remove all tasks from db."""
51
52
           self._db.purge()
       def unique_id(self): # type () -> int
55
            """Return an integer that does not exist in the db."""
           i = 1
56
57
           while self._db.contains(eids=[i]):
58
             i += 1
59
           return i
60
61
       def stop_tasks_db(self):
            """Disconnect from DB."""
63
           pass
```

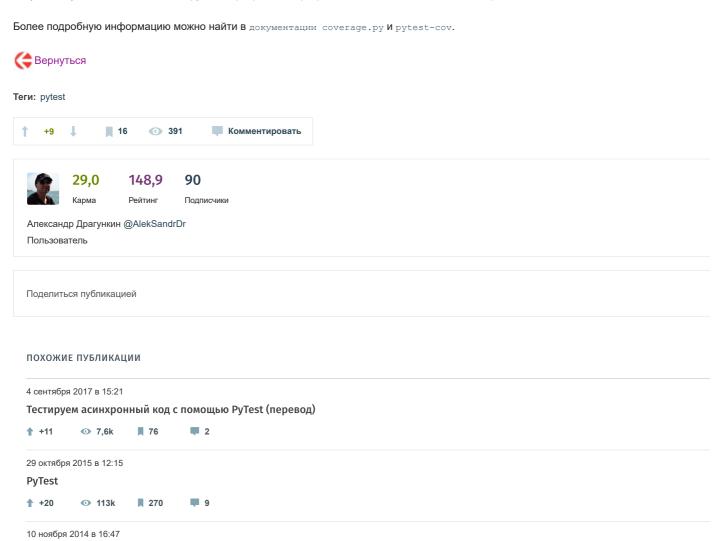
Даже если этот экран не является полной страницей для этого файла, этого достаточно, чтобы сказать нам, что:

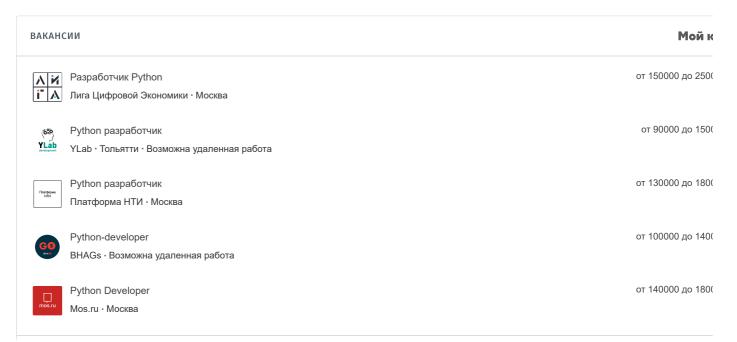
- 1. Мы не тестируем <code>list\_tasks()</code> с установленным владельцем.
- 2. Мы не тестируем update() или delete().

3. Возможно, мы недостаточно тщательно тестируем unique id().

Отлично. Мы можем включить их в наш список ТО-DO по тестированию вместе с тестированием системы конфигурации.

Хотя инструменты покрытия кода чрезвычайно полезны, стремление к 100% покрытию может быть опасным. Когда вы видите код, который тестируется, это может означать, что необходим тест. Но это также может означать, что есть некоторые функции системы, которые не нужн могут быть удалены. Как и все инструменты разработки программного обеспечения, анализ покрытия кода не заменяет мышления.





Как в Яндексе используют PyTest и другие фреймворки для функционального тестирования

**+58** 

**⊙** 70,2k

354

10

Все вакансии

## Комментарии 0

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

#### САМОЕ ЧИТАЕМОЕ



Бунт на Пикабу. Пользователи массово уходят на Реддит

↑ +144 ② 91,5k **■** 71 **■** 290

Смерть курьера «Яндекс.Еды» запустила волну жалоб на условия труда в компании

Как я хакера ловил

Как Мегафон спалился на мобильных подписках

Межпозвоночная грыжа? Работай над ней

Аккаунт	Разделы	Информация	Услуги	Приложения
Войти	Публикации	Правила	Реклама	Загрузите в ДОСТУПНО
Регистрация	Новости	Помощь	Тарифы	Загрузите в App Store Google
	Хабы	Документация	Контент	
	Компании	Соглашение	Семинары	
	Пользователи Конфиденциальность			
	Песочница			
© 2006 – 2019 « <b>TM</b> »	Настройка языка	О сайте Служба поддерж	ки Мобильная версия	