



AlekSandrDr вчера в 17:15

Python Testing с pytest. Конфигурация, ГЛАВА 6

Автор оригинала: Okken Brian

Python

Перевод

Tutorial



В этой главе мы рассмотрим файлы конфигурации, которые влияют на pytest, обсудим, как pytest изменяет свое поведение на их основе, и внесем некоторые изменения в файлы конфигурации проекта Tasks.

The
Pragmatic
Programmers

Python Testing with pytest

Simple, Rapid,
Effective, and
ScalableBrian Okken
edited by Katharine Dvorak

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 примерах. Но что бы следовать вместе с задачами проекта, или адаптировать примеры тестирования для проверки своего собственного проекта (если у вас развязаны!), вы должны перейти на веб-страницу книги и скачать работу. Там же, на веб-странице книги есть ссылка для сообщений об ошибках и дискуссионный форум.

Под спойлером приведен список статей этой серии.

[Оглавление](#)

Конфигурация

До сих пор в этой книге я говорил о различных нетестовых файлах, которые влияют на pytest в основном мимоходом, за исключением config.py который я довольно подробно рассмотрел в главе 5, Плагины, на странице 95. В этой главе мы рассмотрим файлы конфигурации, которые влияют на pytest, обсудим, как pytest изменяет свое поведение на их основе, и внесем некоторые изменения в файлы конфигурации проекта Tasks.

Понимание файлов конфигурации pytest

Прежде чем я расскажу, как вы можете изменить поведение по умолчанию в `pytest`, давайте пробежимся по всем не тестовым файлам в `py` в частности, кто должен заботиться о них.

Следует знать следующее:

- *pytest.ini*: Это основной файл конфигурации Pytest, который позволяет вам изменить поведение по умолчанию. Поскольку вы можете внести довольно много изменений в конфигурацию, большая часть этой главы посвящена настройкам, которые вы можете сделать в `pytest.ini`.
- *conftest.py*: Это локальный плагин, позволяющий подключать хук-функции и фикстуры для каталога, в котором существует файл `conftest.py` и всех его подкаталогов. Файл `conftest.py` описан в главе 5 «Плагины» на стр. 95.
- *__init__.py*: При помещении в каждый `test`-подкаталог этот файл позволяет вам иметь идентичные имена `test`-файлов в нескольких каталогах `test`. Мы рассмотрим пример того, что пойдет не так без файлов `__init__.py` в тестовых каталогах в статье «Избегание коллизий имен файлов» на стр. 120.

Если вы используете `tox`, вас заинтересует:

- *tox.ini*: Этот файл похож на `pytest.ini`, но для `tox`. Однако вы можете разместить здесь свою конфигурацию `pytest` вместо того, чтобы иметь и файл `tox.ini`, и файл `pytest.ini`, сохраняя вам один файл конфигурации. `Tox` рассматривается в главе 7, "Использование `pytest` с инструментами", на стр. 125.

Если вы хотите распространять пакет Python (например, `Tasks`), этот файл будет интересен:

- *setup.cfg*: Это также файл в формате INI, который влияет на поведение файла `setup.py`. Можно добавить несколько строк в `setup.py` для запуска `python setup.py test` и запустить все ваши тесты `pytest`. Если вы распространяете пакет, возможно, у вас уже есть файл `setup.py` и вы можете использовать этот файл для хранения конфигурации Pytest. Вы увидите, как это делается в Приложении 4, «Упаковка и распространение проектов Python», на стр. 175.

Независимо от того, в какой файл вы поместили конфигурацию `pytest`, формат будет в основном одинаковым.

Для `pytest.ini`:

ch6/format/pytest.ini

```
[pytest]
addopts = -rsxX -l --tb=short --strict
xfail_strict = true
... more options ...
```

Для `tox.ini`:

ch6/format/tox.ini

```
... tox specific stuff ...

[pytest]
addopts = -rsxX -l --tb=short --strict
xfail_strict = true
... more options ...
```

Для `setup.cfg`:

ch6/format/setup.cfg

```
... packaging specific stuff ...
[tool:pytest]
addopts = -rsxX -l --tb=short --strict
xfail_strict = true
... more options ...
```

Единственное отличие состоит в том, что заголовок раздела для `setup.cfg` — это `[tool:pytest]` вместо `[pytest]`.

List the Valid ini-file Options with pytest --help

Вы можете получить список всех допустимых параметров для `pytest.ini` из `pytest --help`:

```
$ pytest --help
...
[pytest] ini-options in the first pytest.ini|tox.ini|setup.cfg file found:

markers (linelist)          markers for test functions
empty_parameter_set_mark (string) default marker for empty parametersets
norecursedirs (args)        directory patterns to avoid for recursion
testpaths (args)            directories to search for tests when no files or directories are given in the command line.
console_output_style (string) console output: classic or with additional progress information (classic|progress).
usefixtures (args)          list of default fixtures to be used with this project
python_files (args)         glob-style file patterns for Python test module discovery
python_classes (args)       prefixes or glob names for Python test class discovery
python_functions (args)     prefixes or glob names for Python test function and method discovery
xfail_strict (bool)         default for the strict parameter of xfail markers when not given explicitly (default: False)
junit_suite_name (string)   Test suite name for JUnit report
junit_logging (string)      Write captured log messages to JUnit report: one of no|system-out|system-err
doctest_optionflags (args)  option flags for doctests
doctest_encoding (string)   encoding used for doctest files
cache_dir (string)          cache directory path.
filterwarnings (linelist)   Each line specifies a pattern for warnings.filterwarnings. Processed after -W and --pythonwarn
s.
log_print (bool)            default value for --no-print-logs
log_level (string)          default value for --log-level
log_format (string)         default value for --log-format
log_date_format (string)    default value for --log-date-format
log_cli (bool)              enable log display during test run (also known as "live logging").
log_cli_level (string)      default value for --log-cli-level
log_cli_format (string)     default value for --log-cli-format
log_cli_date_format (string) default value for --log-cli-date-format
log_file (string)           default value for --log-file
log_file_level (string)     default value for --log-file-level
log_file_format (string)    default value for --log-file-format
log_file_date_format (string) default value for --log-file-date-format
addopts (args)              extra command line options
minversion (string)         minimally required pytest version
xvfb_width (string)         Width of the Xvfb display
xvfb_height (string)        Height of the Xvfb display
xvfb_colordepth (string)    Color depth of the Xvfb display
xvfb_args (args)            Additional arguments for Xvfb
xvfb_xauth (bool)           Generate an Xauthority token for Xvfb. Needs xauth.

...
```

Вы увидите все эти настройки в этой главе, за исключением `doctest_optionflags`, который рассматривается в главе 7, "Использование рут другими инструментами", на странице 125.

Плагины могут добавлять опции ini-файлов

Предыдущий список настроек не является константой. Для плагинов (и файлов `conftest.py`) возможно добавить опции файла `ini`. Добавленные опции также будут добавлены в вывод команды `pytest --help`.

Теперь давайте рассмотрим некоторые изменения конфигурации, которые мы можем внести с помощью встроенных настроек INI-файла, доступных в `core pytest`.

Изменение параметров командной строки по умолчанию

Вы использовали уже некоторые параметры командной строки для *pytest*, таких как `-v/--verbose` для подробного вывода `-l/--showlocals` просмотра локальных переменных с трассировкой стека для неудачных тестов. Вы можете обнаружить, что всегда используете некоторые `options-or` и предпочитаете использовать `them-for a project`. Если вы устанавливаете `addopts` в `pytest.ini` для нужных вам параметров, вам больше не придется вводить их. Вот набор, который мне нравится:

```
[pytest]
addopts = -rsxX -l --tb=short --strict
```

Ключ `-rsxX` дает установку *pytest* сообщать о причинах всех `skipped`, `xfailed` или `xpassed` тестов. Ключ `-l` позволит *pytest* вывести трассировку стека для локальных переменных в случае каждого сбоя. `--tb=short` удалит большую часть трассировки стека. Однако, оставит файл и но строки. Параметр `--strict` запрещает использование маркеров, если они не зарегистрированы в файле конфигурации. Вы увидите, как это сделать в следующем разделе.

Регистрация маркеров, чтобы избежать опечаток маркера

Пользовательские маркеры, как описано в разделе «Маркировка тестовых функций» на странице 31, отлично подходят для того, чтобы позволить вам пометить подмножество тестов для запуска определенным маркером. Тем не менее, слишком легко ошибиться в маркере и в конечном итоге некоторые тесты помечены `@pytest.mark.smoke`, а некоторые отмечены `@pytest.mark.somke`. По умолчанию это не ошибка. *pytest* просто думает, что вы создали два маркера. Однако это можно исправить, зарегистрировав маркеры в `pytest.ini`, например так:

```
[pytest]
...
markers =
    smoke: Run the smoke test test functions
    get: Run the test functions that test tasks.get()
...
```

Зарегистрировав эти маркеры, вы теперь также можете увидеть их с помощью `pytest --markers` с их описаниями:

```
$ cd /path/to/code/ch6/b/tasks_proj/tests
$ pytest --markers

@pytest.mark.smoke: Run the smoke test test functions

@pytest.mark.get: Run the test functions that test tasks.get()

@pytest.mark.skip(reason=None): skip the ...

...
```

Если маркеры не зарегистрированы, они не будут отображаться в списке `--markers`. Когда они зарегистрированы, они отображаются в списке, если вы используете `--strict`, любые маркеры с ошибками или незарегистрированные отображаются как ошибки. Единственная разница между `ch6/a/tasks_proj` и `ch6/b/tasks_proj` заключается в содержимом файла `pytest.ini`. В `ch6/a` пусто. Давайте попробуем запустить тесты без регистрации каких-либо маркеров:

```
$ cd /path/to/code/ch6/a/tasks_proj/tests
$ pytest --strict --tb=line

===== test session starts =====

collected 45 items / 2 errors

===== ERRORS =====
_____ ERROR collecting func/test_add.py _____
'smoke' not a registered marker
_____ ERROR collecting func/test_api_exceptions.py _____
'smoke' not a registered marker
!!!!!!!!!!!!!!!!!!!! Interrupted: 2 errors during collection !!!!!!!!!!!!!!!!!!!!!!!
===== 2 error in 1.10 seconds =====
```

If you use markers in `pytest.ini` to register your markers, you may as well add `--strict` to your `addopts` while you're at it. You'll thank me later. Let's ahead and add a `pytest.ini` file to the `tasks` project:

Если вы используете маркеры в `pytest.ini` для регистрации своих маркеров, вы также можете добавить `--strict` к своим `addopts`. Ты поблагодаришь меня позже. Давайте продолжим и добавим файл `pytest.ini` в проект `tasks`:

Если вы используете маркеры в `pytest.ini` для регистрации маркеров, вы можете также добавить `--strict` к имеющимся при помощи `addopts`. Круто?! Отложим благодарности и добавим файл `pytest.ini` в проект `tasks`:

`ch6/b/tasks_proj/tests/pytest.ini`

```
[pytest]
addopts = -rsxX -l --tb=short --strict
markers =
    smoke: Run the smoke test test functions
    get: Run the test functions that test tasks.get()
```

Здесь комбинация флагов предпочитаемые по умолчанию:

- `-rsxX`, чтобы сообщить, какие тесты `skipped`, `xfailed`, или `xpassed`,
 - `--tb = short` для более короткой трассировки при сбоях,
 - `--strict` что бы разрешить только объявленные маркеры.
- И список маркеров для проекта.

Это должно позволить нам проводить тесты, в том числе дымовые(smoke tests):

```
$ cd /path/to/code/ch6/b/tasks_proj/tests
$ pytest --strict -m smoke

===== test session starts =====
collected 57 items

func/test_add.py .
func/test_api_exceptions.py ..

===== 54 tests deselected =====
===== 3 passed, 54 deselected in 0.06 seconds =====
```

Требование минимальной версии Pytest

Параметр `minversion` позволяет указать минимальную версию `pytest`, ожидаемую для тестов. Например, я задумал использовать `approx()` в тестировании чисел с плавающей запятой для определения "достаточно близкого" равенства в тестах. Но эта функция не была введена в `pytest` до версии 3.0. Чтобы избежать путаницы, я добавляю следующее в проекты, которые используют `approx()`:

```
[pytest]
minversion = 3.0
```

Таким образом, если кто-то пытается запустить тесты, используя более старую версию `pytest`, появится сообщение об ошибке.

Остановка pytest от поиска в неправильных местах

Знаете ли вы, что одно из определений «norecuse» заключается в том, что бы дважды выругаться в собственном коде? Ну, нет. На самом деле это означает учет подкаталогов. `pytest` включает обнаружение тестов рекурсивно исследуя кучу каталогов. Но есть некоторые каталоги, которые вы хотите исключить из просмотра `pytest`.

Значением по умолчанию для `norecuse` является `'.' * Build dist CVS _dargs {arch} and *.egg. Having '.*' — это хорошая причина не включать вашу виртуальную среду '.venv', потому что все каталоги, начинающиеся с точки, не будут видны.`

В случае проекта `Tasks`, не помешает указать `src`, потому что поиск в тестовых файлах с помощью `pytest` будет пустой тратой времени.

```
[pytest]
norecursedirs = .* venv src *.egg dist build
```

При переопределении параметра, который уже имеет полезное значение, такого как этот параметр, полезно знать, какие есть значения по умолчанию, и вернуть те, которые вам нужны, как я делал в предыдущем коде с `*.egg dist build`.

`norecursedirs` — своего рода следствие для тестовых путей, поэтому давайте посмотрим на это позже.

спецификация дерева тестового каталога

В то время как `norecursedirs` указывает `pytest` куда не надо заглядывать, `testpaths` говорит `pytest`, где искать. `testpaths` — это список каталогов относительно корневого каталога для поиска тестов. Он используется только в том случае, если в качестве аргумента не указан каталог, файл или `nodeid`.

Предположим, что для проекта `Tasks` мы поместили `pytest.ini` в каталог `tasks_proj` вместо тестов:

```
\code\tasks_proj>tree/f
.
|
|_ pytest.ini
|_ src
|   |_ tasks
|       api.py
|       ...
|_ tests
|   |_ conftest.py
|   |_ pytest.ini
|   |_ func
|       test_add.py
|       ...
|   |_ unit
|       test_task.py
|       __init__.py
|       ...
```

Тогда может иметь смысл поместить тесты в `testpaths`:

```
[pytest]
testpaths = tests
```

Теперь, если вы запускаете `pytest` из каталога `tasks_proj`, `pytest` будет искать только в `tasks_proj/tests`. Проблема здесь в том, что во время разработки и отладки тестов я часто перебираю тестовый каталог, поэтому я могу легко тестировать подкаталог или файл, не указывая вес. Поэтому мне этот параметр мало помогает в интерактивном тестировании.

Тем не менее, он отлично подходит для тестов, запускаемых с сервера непрерывной интеграции или с `tox`-а. В этих случаях вы знаете, что корневой каталог будет фиксированным, и вы можете перечислить каталоги относительно этого фиксированного корневого каталога. Это те случаи, когда вы действительно хотите сократить время тестирования, так что избавиться от поиска тестов — это здорово.

На первый взгляд может показаться глупым использовать одновременно и тестовые пути, и `norecursedirs`. Однако, как вы уже видели, тестовые пути мало помогают в интерактивном тестировании из разных частей файловой системы. В этих случаях `norecursedirs` могут помочь. Кроме того, если у вас есть каталоги с тестами, которые не содержат тестов, вы можете использовать `norecursedirs`, чтобы избежать их. Но на самом деле, какой смысл ставить дополнительные каталоги в тесты, которые не имеют тестов?

Изменение Правил Обнаружения Тестов

`pytest` находит тесты для запуска на основе определенных правил обнаружения тестов. Стандартные правила обнаружения тестов:

- Начните с одного или нескольких каталогов. Вы можете указать имена файлов или каталогов в командной строке. Если вы ничего не указываете, используется текущий каталог.

- Искать в каталоге и во всех его подкаталогах тестовые модули.
- Тестовый модуль — это файл с именем, похожим на `test_*.py` или `*_test.py`.
- Посмотрите в тестовых модулях функции, которые начинаются с `test`.
- *Ищите классы, которые начинаются с `Test`. Ищите методы в тех классах, которые начинаются с ``test``, но не имеют метода `__init__`.*

Это стандартные правила обнаружения; Однако вы можете изменить их.

python_classes

Обычное правило обнаружения тестов для `pytest` и классов — считать класс потенциальным тестовым классом, если он начинается с `Test`. Класс также не может иметь метод `__init__()`. Но что, если мы захотим назвать наши тестовые классы как `<something>Test` или `<something>Suite`? Вот где приходит `python_classes`:

```
[pytest]
python_classes = *Test Test* *Suite
```

Это позволяет нам называть классы так:

```
class DeleteSuite():

    def test_delete_1():
        ...

    def test_delete_2():
        ...

    ....
```

python_files

Как и `pytest_classes`, `python_files` изменяет правило обнаружения тестов по умолчанию, которое заключается в поиске файлов, начинающихся с `test_*` или имеющих в конце `*_test`.

Допустим, у вас есть пользовательский тестовый фреймворк, в котором вы назвали все свои тестовые файлы `check_<something>.py`. Кажется разумным. Вместо того, чтобы переименовывать все ваши файлы, просто добавьте строку в `pytest.ini` следующим образом:

```
[pytest]
python_files = test_* *_test check_*
```

Очень просто. Теперь вы можете постепенно перенести соглашение об именах, если хотите, или просто оставить его как `check_*`.

python_functions

`python_functions` действует как две предыдущие настройки, но для тестовых функций и имен методов. Значение по умолчанию — `test_*`. Чтобы добавить `check_*`—вы угадали—сделайте это:

```
[pytest]
python_functions = test_* check_*
```

Соглашения об именах `pytest` не кажутся такими уж ограничивающими, не так ли? Так что, если вам не нравится соглашение об именах по умолчанию, просто измените его. Тем не менее, я призываю вас иметь более вескую причину для таких решений. Миграция сотен тестовых файлов — определенно веская причина.

Запрет XPASS

Установка `xfail_strict = true` приводит к тому, что тесты, помеченные `@pytest.mark.xfail`, не распознаются, как вызвавшие ошибку. Я думаю, что эта установка должно быть всегда. Дополнительные сведения о маркере `xfail` см. В разделе "Маркировка тестов ожидающих сбоя" на 37.

Предотвращение конфликтов имен файлов

Полезность наличия файла `__init__.py` в каждом тестовом подкаталоге проекта долго меня смущали. Однако разница между тем, чтобы их или не иметь, проста. Если у вас есть файлы `__init__.py` во всех ваших тестовых подкаталогах, вы можете иметь одно и то же тестовое имя файла в нескольких каталогах. А если нет, то так сделать не получится.

Вот пример. Каталог `a` и `b` оба имеют файл `test_foo.py`. Неважно, что эти файлы содержат в себе, но для этого примера они выглядят так:

ch6/dups/a/test_foo.py

```
def test_a():
    pass
```

ch6/dups/b/test_foo.py

```
def test_b():
    pass
```

С такой структурой каталогов:

```
dups
├── a
│   └── test_foo.py
└── b
    └── test_foo.py
```

Эти файлы даже не имеют того же контента, но тесты испорчены. Запускать их по отдельности получится, а запустить `pytest` из каталога `dups` не получится:

```
$ cd /path/to/code/ch6/dups
$ pytest a
===== test session starts =====

collected 1 item

a\test_foo.py .

===== 1 passed in 0.05 seconds =====

$ pytest b
===== test session starts =====

collected 1 item

b\test_foo.py .

===== 1 passed in 0.05 seconds =====

$ pytest
===== test session starts =====

collected 1 item / 1 errors

===== ERRORS =====
_____ ERROR collecting b/test_foo.py _____
import file mismatch:
imported module 'test_foo' has this __file__ attribute:
  /path/to/code/ch6/dups/a/test_foo.py
which is not the same as the test file we want to collect:
  /path/to/code/ch6/dups/b/test_foo.py
HINT: remove __pycache__ / .pyc files and/or use a unique basename for your test file modules
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 errors during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.34 seconds =====
```


Ни чего не понятно!

Это сообщение об ошибке не дает понять, что пошло не так.

Чтобы исправить этот тест, просто добавьте пустой `__init__.py` файл в подкаталоги. Вот пример каталога `dups_fixed` такими же дублиров: именами файлов, но с добавленными файлами `__init__.py`:

```
dups_fixed/
├── a
│   ├── __init__.py
│   └── test_foo.py
└── b
    ├── __init__.py
    └── test_foo.py
```

Теперь давайте попробуем еще раз с верхнего уровня в `dups_fixed`:

```
$ cd /path/to/code/ch6/ch6/dups_fixed/

$ pytest
===== test session starts =====

collected 2 items

a\test_foo.py .
b\test_foo.py .

===== 2 passed in 0.15 seconds =====
```

Так то будет лучше.

Вы, конечно, можете убеждать себя, что у вас никогда не будет повторяющихся имен файлов, поэтому это не имеет значения. Все, типа, нормально. Но проекты растут и тестовые каталоги растут, и вы точно хотите дождаться, когда это случится с вами, прежде чем позаботи этом? Я говорю, просто положите эти файлы туда. Сделайте это привычкой и не беспокойтесь об этом снова.

Упражнения

In Chapter 5, Plugins, on page 95, you created a plugin called `pytest-nice` that included a `--nice` command-line option. Let's extend that to include a `pytest.ini` option called `nice`.

В главе 5 «Плагины» на стр. 95 вы создали плагин с именем `pytest-nice` который включает параметр командной строки `--nice`. Давайте расширим это, включив опцию `pytest.ini` под названием `nice`.

1. Добавьте следующую строку в хук-функцию `pytest_addoption` `pytest_nice.py`: `parser.addini('nice', type='bool', help='Turn fail into opportunities.')`
2. Места в плагине, которые используют `getoption()`, также должны будут вызывать `getini('nice')`. Сделайте эти изменения.
3. Проверьте это вручную, добавив `nice` в файл `pytest.ini`.
4. Не забудьте про тесты плагинов. Добавьте тест, чтобы убедиться, что параметр `nice` из `pytest.ini` работает корректно.
5. Добавьте тесты в каталог плагинов. Вам нужно найти некоторые дополнительные функции Pytester.

Что дальше

В то время как `pytest` является чрезвычайно мощным сам по себе—особенно с плагинами—он также хорошо интегрируется с другими инструментами разработки программного обеспечения и тестирования программного обеспечения. В следующей главе мы рассмотрим использование `pytest` в сочетании с другими мощными инструментами тестирования.



Теги: `pytest`

↑

+6

↓

🔖


21

👁

942

💬

2



29,0

Карма

148,9

Рейтинг

90

Подписчики

Александр Драгункин

@AleksandrDr

Пользователь

Поделиться публикацией

- ПОХОЖИЕ ПУБЛИКАЦИИ
- 4 сентября 2017 в 15:21

Тестируем асинхронный код с помощью PyTest (перевод)

↑

+11

👁

7,6k

🔖

76

💬

2
- 29 октября 2015 в 12:15

PyTest

↑

+20

👁

113k

🔖

270

💬

9
- 10 ноября 2014 в 16:47

Как в Яндексе используют PyTest и другие фреймворки для функционального тестирования

↑

+58

👁

70,2k

🔖


354

💬

10

ЗАКАЗЫ	Фриланс
Разработчик ПО машинного зрения	?
6 откликов · 62 просмотра	300000
Разработка WHATSAPP бота	100000
11 откликов · 65 просмотров	за пр
Ищу напарника на react native для совместной работы	100000
5 откликов · 76 просмотров	за пр
Разработка модуля поискового механизма (Elasticsearch)	500000
9 откликов · 49 просмотров	за пр
Сделать небольшое веб приложение fullstack	300000
26 откликов · 131 просмотр	за пр
Все заказы	Разместить заказ

Комментарии 2

- 

kekekeks

вчера в 17:30

🔖

🔖

Горшочек, не вари!

↑
- 

AleksandrDr

вчера в 17:48

🔖

🔖

🔖

🔖

Извиняйте! Газ кончился. Горшок остывает.

↑

Только полноправные пользователи могут оставлять комментарии. [Войдите, пожалуйста.](#)

САМОЕ ЧИТАЕМОЕ

- Сутки
- Неделя
- Месяц

Бунт на Пикабу. Пользователи массово уходят на Реддит

+144 91,5k 71 290

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

+57 57,1k 12 352

Как я хакера ловил

+106 17,6k 81 49

Как Мегафон спалился на мобильных подписках

+500 89,2k 174 462

Межпозвоночная грыжа? Работай над ней

+37 20,6k 163 51

Аккаунт	Разделы	Информация	Услуги	Приложения
Войти	Публикации	Правила	Реклама	<div> Загрузите в App Store</div> <div> доступно Google</div>
Регистрация	Новости	Помощь	Тарифы	
	Хабы	Документация	Контент	
	Компании	Соглашение	Семинары	
	Пользователи	Конфиденциальность		
	Песочница			