Přechod webových aplikací na Python 3

Tomáš Pazderka ▪ tomas.pazderka@nic.cz ▪ 6. listopadu 2018

Python 3

- První verze vyšla 2008
- První “production ready” verze vyšla 2009
- Řešení některých chyb v návrhu jazyka
- Změna nekompatibilní s python 2
- Reorganizace standardních knihoven

Nejčastější rozdíly oproti python 2

Problém

- print
- xrange
- dělení
- `iter` metody
- přesun/přejmenovaní modulu
- text vs. bytes

Řešení

- přepsat na funkci
- `from builtins import range`
- `//` pro celočíselné dělení
- `from builtins import ...`
- detekce verze, wrapper (six)
- zadefinovat API

Proč přejít?

- $3 > 2$
  - Modernější jazyk
  - Aktivní vývoj
  - Rozšíření stdlib
  - AsyncIO
- Konec podpory pro Python 2.7 v lednu 2020

Dostupné pomocné nástroje

- 2to3 → konverze python 2 na python 3
- futurize → konverze python 2 na python 2/3
- modernize → konverze python 2 na python 2/3
- six → wrapper pro kompatibilitu 2/3
- future → backport python 3 konstrukcí do python 2
- past → implementace python 2 konstrukcí v python 3

Předpoklady pro úspěšnou portaci

- Závislosti
  - 349/360 nejpopulárnějších aplikací má podporu pro Python 3
  - caniusepython3
  - Výměna za jinou knihovnu, fork
- Testujte
  - Dobré pokrytí testy výrazně usnadní práci
  - Pomůže odhalit nekompatibility tam, kde není zřejmé rozhraní
- Refaktoring
  - Odstranit nepoužitý kód
  - Odstranit zastaralé metody

Naše codebase

- 29 projektů, ~ 100k LOC
- Vzájemné závislosti, někdy i poměrně složité
- Cca 50 externích závislostí
- Některé z nich opuštěné a nespravované

Naší jedinou volbou byl převod na Python 2/3

Použité knihovny

- isort – automatické řazení importů
- autopep8 – nástroj pro formátování kódu
- tox – framework pro správu testovacích prostředí
- six – knihovna zajišťující kompatibilitu pro Py2 a Py3

isort & autopep8

- isort
  - hromadné přidání importů

    ```
    from __future__ import unicode_literals
    ```
  - usnadnění zafixování str/bytes API
- autopep8
  - normalizace indentace a mezer
  - odstranění zastaralých metod (`has_key`)
  - odchycení výjimek
  - porovnávání typů a další ...

Tox

- Projekt pro automatizaci činností pro python projekty
- Umožňuje snadno a parametricky vytvářet virtuální prostředí
  - volitelné závislosti
  - různé verze pythonu
  - různé verze knihoven
- Snadná konfigurace
- Sjednocení procesů pro vývojové prostředí a CI

Definice dostupných prostředí

```
[tox]
envlist =
    quality
    clear-coverage
    py27-{pure, dep}
    {py35, py36, py37}-{pure, dep}
    compute-coverage
skip_missing_interpreters = True
```

Společná nastavení

```
[testenv]
setenv =
    PYTHONPATH = {toxinidir}/test_cfg:{env:IDL_DIR:}
    DJANGO_SETTINGS_MODULE = settings
passenv =
    PG*
    CI*
skip_install =
    coverage: True
extras =
    testing
    # install dependencies from setup.py extras_require['dependency']
    dep: dependency
deps =
    coverage
install_command = pip install --process-dependency-links {opts} {packages}
commands =
    coverage run --parallel-mode --source=APP --branch -m django test APP
```

Nastavení konkrétního prostředí

```
[testenv:clear-coverage]
commands = coverage erase

[testenv:compute-coverage]
commands =
    coverage combine
    coverage report --include=*/tests/*  --fail-under=100 --show-missing
    coverage report --omit=*/tests/*

[testenv:quality]
basepython = python2.7
whitelist_externals = msgcmp
extras = quality
ignore_errors = True # Do not fail on first error, but run all the checks
commands =
    isort --recursive --check-only --diff APP
    django-admin makemessages --locale C --no-obsolete --keep-pot
    msgcmp APP/locale/cs/LC_MESSAGES/django.po APP/locale/django.pot
```

Převod projektu na python 3 kompatibilní s pomocí knihovny `six`

- Nahrazení typů `unicode` a `str` za `six.text_type` a `six.binary_type`
- Definice `__str__` a `__unicode__`
  - definovat `__str__` a třídu obalit `@six.python_2_unicode_compatible`
- Definice `__nonzero__`, nově `__bool__` – doporučeno implementace svázat
- `import urlparse/urllib` nahradit za
  `import six.moves.urllib.parse`
- `StringIO/BytesIO` importovat ze `six`
- Důsledně používat *binární* mód při otevírání binárních souborů

Perličky

- Podivná API - *str* v obou verzích pythonu

```
if isinstance(name, six.text_type) and six.PY2:
    name = name.encode()
elif isinstance(name, six.binary_type) and six.PY3:
    name = name.decode()
```

- Porovnávání různých typů

```
test = None
if 2 < test:
    do something
```

Jak psát kompatibilní kód

- Python 2 by měl být výjimkou ...

```
if six.PY2:
    handle python 2
else:
    handle python 3
```

- Konverze mezi textem a bytes by měla probíhat na hranicích kódu
- Vyhnout se `iteritems` pokud to není nezbytné pro výkon
  - pokud ano, tak `six.iteritems`
- Zadefinovat API (unicode/bytes)

Děkuji za pozornost

. . . a zadefinujte si API!