

Django 3000

Martin v. Löwis

Überblick

- Python-3-Support in Django voraussichtlich ab Django 1.5
- bis dahin: Subversion-Branch
- Portierungsideen
- Django-Portierung
- Portierung von Anwendungen

Python-3-Portierung

- Mehrere Alternativen
- "Burn your bridges": keine 2.x-Unterstützung mehr
- "Single Source": gleiche Quellen für Python 2 und Python 3; nicht notwendigerweise gleicher Code zur Laufzeit
- identische Quellen
- Prerequisites!

Burn Your Bridges

- 2to3: Tool für automatische Konvertierung
- einmalig aufrufen, Konvertierungsfehler beheben
- Variante: 2 Versionen parallel gepflegt
- akzeptabel für Endanwendungen / Services, nicht akzeptabel für Bibliotheken (wie Django)

Single Source

- Ziel: 2to3 erzeugt Code, der dann unverändert auf Python 3 läuft
- Originalcode soll auf Python 2 laufen
- Notwendige Zusatzänderungen müssen sowohl auf Python 2 und 3 laufen
- wenn möglich: Integration von 2to3 in setup.py
- `python3 setup.py build`
- Variante: Originalcode ist Python 3, Verwendung von 3to2

Identische Quellen

- Formulierung von Code so, dass er sowohl unter Python 2 als auch unter Python 3 läuft
- schwierig wegen syntaktischer Unterschiede
 - `try: ... except Exception, e:`
 - `u"String"`
- einfacher, falls mindestens Python 2.6 vorausgesetzt werden kann
- Hilfe durch Benjamin Petersons "six"

Django-Portierung

Quellen

- <https://bitbucket.org/loewis/django-3k>
- <http://code.djangoproject.com/svn/django/branches/features/py3k/>
- <https://code.djangoproject.com/browser/django/branches/features/py3k>

setup.py

- distutils
- Grundidee:
`cmdclasses = { 'build_py':build_py_2to3 }`
- Django: Tests müssen auch mit 2to3 konvertiert werden
 - einschließlich der doctests

2to3-Probleme

- `gvqs = Location.objects.values()`
wird zu
`gvqs = list(Location.objects.values())`
- Original-Code liefert `QuerySet`;
transformierter Code führt `Query` aus
- Lösung: `.values-Fixer` unterdrücken

2to3-Probleme (2)

- an manchen Stellen erwartet Django, dass `.values()` eine Liste liefert
- `for arg in list(args) + kwargs.values():`
- Lösung: `dictvalues-Helper-Funktion`
- `for arg in list(args) + dictvalues(kwargs)`

2to3-Probleme (3)

- `if isinstance(o, basestring):` # Original
- `if isinstance(o, str):` # 2to3
- Lösungsversuch
`if isinstance(o, (str, unicode)):`
- `if isinstance(o, (str, str)):` # 2to3
- Lösung:
`if isinstance(o, (bytes, unicode)):`

2to3-Korrekturen

- `print()`
- `try-except`
- `list(d.items()); .iteritems -> .items`
- Verzicht auf `u`-Präfix
- `callable(X) -> isinstance(X, collections.Callable)`
- Imports: `urlparse -> urllib.parse, ...`
- relative Imports
- ...

Manuelle Korrekturen

- `django.utils.py3`: Hilfsfunktionen
- Hauptänderungen: Bytes-Literale
 - 2to3 ändert automatisch `u"foo"` -> `"foo"`
 - Änderung `"foo"` -> `b"foo"` nicht praktikabel
 - Hilfsfunktion `b("foo")`, um Bytes-Literale zu deklarieren

b()

- ```
if sys.version_info < (3,):
 def b(s):
 return s
else:
 def b(s):
 # vereinfacht
 return s.encode("ascii")
```

# bytes vs. str

- Festlegung von Bytes vs. Unicode
- bereits intensive Verwendung von Unicode in Django
- Design-Entscheidung: "Protokoll-Strings" sind Bytes (HTTP-Fragmente, MIME-Fragmente, ...)
- Ausnahmen: HTTP-Headernamen
  - `req.content_type = response['Content-Type']`

# bytes vs. str

- Django: `smart_str`, `smart_unicode`
  - liefern stets `bytes` / `unicode`, für 2.x und 3.x
- Py3: `smart_text`
  - liefert Text-Typ der jeweiligen Version
  - Rückgabetyt von `__repr__`
  - Schlüsselwortargumente
  - Parsen von `datetime` im DB-Adapter

# Manuelle Korrekturen (2)

- StringIO: `io.BytesIO` oder `io.StringIO`?
- 2to3 ändert Imports auf StringIO

# Manuelle Korrekturen (3)

- ```
def keys(self):      # Original  
    return list(self.iterkeys())
```
- ```
def keys(self): # 2to3
 return list(self.keys())
```
- ```
def keys(self):      # work-around  
    return list(getattr(self, "iterkeys"))
```

Manuelle Korrekturen (4)

- ```
def __str__(self): # Original
 return self.__unicode__().encode('utf-8')
```
- ```
if sys.version_info < (3,):
    def __str__(self): # siehe oben
else:
    def __str__(self):
        return self.__unicode__()
```

Manuelle Korrekturen (5)

- doctests: Django-lokale Kopie
- Problem: Mit 3.2 tauchten extra \0-Zeichen in Ausgabe auf
 - "foo" != "foo"
- Ursache: StringIO.truncate beläßt File-Position
 - zusätzliches .seek nötig

Anwendungsportierung

django-debug-toolbar

jQuery Test

If you see this, jQuery **1.2.6** is working.

Hide »

Versions

DJANGO 1.4 PRE-ALPHA

Time

CPU: 1.47ms (1.47ms)

Settings

HTTP Headers

Request Vars

SQL

0 QUERIES IN 0.00ms

Templates

Signals

Logging

0 MESSAGES

Schritt 1

- Abhängigkeiten
- django-debug-toolbar: keine

Schritt 2

- Portierung auf Django ~~1.5~~ 1.4
- settings.py:
 - DATABASES statt DATABASE_ENGINE
 - MEDIA_URL mit / terminieren
 - django.contrib.auth.context_processors.auth statt django.core.context_processors.auth

Schritt 3

- Portierungsstrategie wählen
- django-debug-toolbar:
 - identische Quellen für 2.x und 3.x
 - Unterstützung von Python-Versionen vor 2.6

Unicode-Literale

- `from django.utils.py3 import u`
- `u'foo'` durch `u('foo')` ersetzen

print

- `print "foo"` durch `print("foo")` ersetzen
- `print` durch `print()` ersetzen
- hier: keine Anwendungen von `print` mit mehreren Argumenten

SocketServer

- try:
 import SocketServer
except ImportError:
 import socketserver as SocketServer

keys

- `keys = self.signals.keys()`
`keys.sort()`
- Ersetzen durch
`keys = list(self.signals.keys())`
`keys.sort()`
- Alternativ:
`keys = sorted(self.signals.keys())`
ab Python 2.4

itervalues()

- ... for d in self._databases.itervalues()
- durch .values() ersetzen

raise

- `raise ImproperlyConfigured, "text"`
- ersetzen durch
`raise ImproperlyConfigured("text")`

except

- `except ImportError, e:`
...
- `except ImportError:`
 `t, e, tb = sys.exc_info()`
...

Fragen?