

Portierung von ZODB auf Python 3.1

Martin v. Löwis

Überblick

- Prototyp eines ZODB-Ports auf Python 3
 - ZODB: Zope Object Database
- Portierungsstrategie
- Portierte Pakete
- Probleme

<http://tinyurl.com/py3zodb>

Python 3

- Python 3.0: 3. Dezember 2008
- Python 3.1: 27. Juni 2009
- What's new in Python 3000:
 - print ist eine Funktion (exec auch)
 - Dictionary-Methoden liefern Views
 - Beliebige Objekte nicht mehr total geordnet
 - Kein int mehr (int is long); true division
 - Strings sind Unicode, bytes, bytearray
 - ...

<http://tinyurl.com/py3zodb>

Portierungsstrategie

- Gleicher Quelltext für Python 2 und Python 3
- möglichst keine Auswirkungen auf Verhalten unter Python 2
- Python-Code: 2to3
 - Integration in Build-Prozess (setup.py build)
- C-Code: Präprozessormagie
 - möglichst wenig bedingten Code

<http://tinyurl.com/py3zodb>

2to3

2to3

- Guido van Rossum, Colin Winter, Benjamin Peterson
- Konvertiert 2.x-Code statisch in 3.x
 - zuverlässig für syntaktische Umformulierungen
 - Heuristiken für Bibliotheksänderungen
 - Erweiterbar: muster-basierte "fixer"
 - relativ langsam

<http://tinyurl.com/py3zodb>

2to3-Beispiel: Ausnahmen

```
except IOError, e:
```

➔ `except IOError as e:`

```
raise t, v, tb
```

➔ `raise t(v).with_traceback(tb)`

```
class POSError(StandardError):
```

➔ `class POSError(Exception):`

<http://tinyurl.com/py3zodb>

2to3-Beispiel: builtins

```
self.assert_(self.tree.has_key(k))
```

➔

```
self.assert_(k in self.tree)
```

```
items=base.items()
```

➔

```
items=list(base.items())
```

```
set(range(11, 50)+range(106, 110)+range(200, 305))
```

➔

```
set(list(range(11, 50))+list(range(106, 110))  
+list(range(200, 305)))
```

```
isinstance(addr[1], types.IntType)
```

➔

```
isinstance(addr[1], int)
```

<http://tinyurl.com/py3zodb>

2to3: Standardbibliothek

```
import copy_reg  
copy_reg.constructor(simple_new)
```

➔

```
import copyreg  
copyreg.constructor(simple_new)
```

```
from UserList import UserList
```

➔

```
from collections import UserList
```

```
import cPickle as pickle
```

➔

```
import pickle as pickle
```

```
reduce(...)
```

➔

```
from functools import reduce
```

<http://tinyurl.com/py3zodb>

lib2to3-Bug

```
def replace(self, new):
    assert self.parent is not None
    if not isinstance(new, list):
        new = [new]
    # erzeuge neue children-Liste für self.parent
    L_children = ...
    self.parent.children = L_children
    for x in new:
        x.parent = self.parent
    self.parent = None
```

<http://tinyurl.com/py3zodb>

Portierte Module

setuptools

- Entwickelt von Phillip Eby (PJE)
 - Erste Portierung von Lennart Regebro
 - "reine" Python-3-Portierung, für 3.0
 - 3.1: Tarek Ziadés distutils-Änderungen brechen setuptools
 - unglückliche Rekursion von egg_info und sdist
 - build_ext.get_ext_filename auch mit unqualifiziertem Namen gerufen
 - Unterstützung für 2to3
- <http://tinyurl.com/py3zodb>

setuptools: build_ext

- `get_ext_filename`: Convert the name of an extension (eg. "foo.bar") into the name of the file from which it will be loaded (eg. "foo/bar.so", or "foo\bar.pyd")
- Überschrieben von setuptools:
 - falls name ein Library-Objekt bezeichnet: bar.a
 - falls ein anderes Extension-Modul eingebunden wird (für Unix und OSX: dl-bar.so)
 - Implementierungsstrategie: dictionary der gültigen Extensionnamen
- Work-around: trage auch unqualifizierten Namen in map ein

<http://tinyurl.com/py3zodb>

setuptools: 2to3-Integration

- Neue Option für `build_py`: `--with-2to3`
- Keine Auswirkung unter Python 2
- anzugeben am besten in `setup.py`
 - `options={'build_py': {'with_2to3': True}}`
- TODO: Konfiguration von Fixern (Ausschluss von schlechten Fixern, Hinzunahme weiterer Fixer)

<http://tinyurl.com/py3zodb>

zope.interfaces

- Initialer Port von Lennart Regebro
- Hauptproblem: `__metaclass__` wird nicht mehr berücksichtigt
- neu: Portierung des C-Moduls

<http://tinyurl.com/py3zodb>

zope.interfaces: Fixer

```
class _Family(object):  
    zope.interface.implements(BTrees.Interfaces.IBTreeFamily)  
    ...
```

```
@zope.interface.implementer(BTrees.Interfaces.IBTreeFamily)  
class _Family(object):  
    ...
```

- Automatisiert in Paket `zope.fixers`
 - Bug: erkennt `implements()` und `zope.interface.implements`, aber nicht `interface.implements()`
 - lib2to3-API-Änderung von 3.0 nach 3.1

<http://tinyurl.com/py3zodb>

transaction

- reines Python
- einzige inhaltliche Änderung:
 - `L.sort(rm_cmp)`
 - `rm_cmp: cmp(rm1.sortKey(), rm2.sortKey())`
 - `L.sort(key=lambda rm:rm.sortKey())`

<http://tinyurl.com/py3zodb>

zope.proxy

- Zweck: delegiert alle API-Rufe auf ein anderes Objekt
- C
- Portierung prototypisch
- gelöschte Methoden (z.B. nb_long) werden beim Kompilieren weggelassen
- neue Methoden werden aber bisher nicht delegiert

<http://tinyurl.com/py3zodb>

Sonstige Pakete

- "Portiert" durch Einfügen der 2to3-Option
- `zc.lockfile`
- `zdaemon`
- `ZConfig`
- `zope.event`

<http://tinyurl.com/py3zodb>

ZODB

- C und Python
- Python-Code: im wesentlichen über 2to3 portiert
- Abhängigkeiten (install_requires) zunächst deaktiviert, da svn-Versionen der portierten Pakete nicht als aktuell gelten
- z.B. hat svn transaction Version 0, setuptools versucht, "neuere" Version zu installieren

<http://tinyurl.com/py3zodb>

Probleme: Strings

Strings

- String-Literale ergeben in Py3k Unicode-Objekte (Typ str)
- Klassennamen, Variablennamen usw. ebenfalls durch Unicode-Strings repräsentiert
- keine automatische Konvertierung in Bytes
- io unterscheidet zwischen Textmodus und Binärmodus (auch für Unix)
- Bei jeder Verwendung von Strings ist zu überlegen, ob es sich inhaltlich um Text oder Binärdaten handelt

<http://tinyurl.com/py3zodb>

Strings(2)

- Neuer Typ `bytes` zur Speicherung von Bytefolgen
- Keine automatische Konvertierung in `str`;
explizite Konvertierung benötigt `encoding`
 - `str(bytesobject)` äquivalent zu `repr()`, `b"data"`
- `io`: 2 Versionen jeder Klasse
 - `io.StringIO`: parallele Klasse `io.BytesIO`

<http://tinyurl.com/py3zodb>

2to3

- 2to3 nimmt i.d.R. an, dass Strings im 2.x Quelltext als Strings gemeint sind
- 2.6-Feature: bytes-Literale erlaubt
- unicode wird durch str ersetzt, basestring auch

<http://tinyurl.com/py3zodb>

Byte-Operationen

- Zahlreiche ZODB-Daten sind konzeptionell Bytes:
 - Magic Codes (FS21, \0BLOBSTART, ...)
 - Transaction IDs
 - Object IDs
- wird an zahlreichen Stellen aber als String im Quelltext notiert
- StringIO wird in ZODB als Byte-Stream verwendet

<http://tinyurl.com/py3zodb>

Byte-Operationen: Lösung

- Neue Funktion `b(s)`: `z64 = b('\0'*8)`
- Identität für 2.x
- `s.encode('latin-1')` für 3.x
- `as_text: bytes->str`
- Verwendet für Transactionsstatus (in 'ucp')

<http://tinyurl.com/py3zodb>

Byte-Operationen: Lösung

```
from cStringIO import StringIO
```

➔

```
from io import StringIO
```

```
if sys.version_info <= (3,):
```

```
    from cStringIO import StringIO
```

```
else:
```

```
    from io import BytesIO as StringIO
```

➔

```
if sys.version_info <= (3,):
```

```
    from io import StringIO
```

```
else:
```

```
    from io import BytesIO as StringIO
```

<http://tinyurl.com/py3zodb>

C

- PyString_* existiert nicht mehr, nur PyUnicode_*, PyBytes*
- Strings als Text: definiere neue Makros
 - PyText_FromString (2.x: PyString_FromString, 3.x PyUnicode_FromString), PyText_Check
 - Vorsicht, falls PyUnicode_Check bereits berücksichtigt
 - char* -> PyUnicode nimmt UTF-8 an
- Strings als Bytes: analog

<http://tinyurl.com/py3zodb>

C

- Extraktion von `char*` aus Unicode-Objekt
- Problem: Speicherverwaltung (wo kommt der Speicher her)
- Problem: Encoding (z.B. fest UTF-8)
- Problem: Effizienz

<http://tinyurl.com/py3zodb>

C

- Speicherverwaltung: Konvertiere PyUnicode erst in PyBytes, und extrahiere dann char*
- encode-Operation kann scheitern
- PyUnicode_AsUTF8String

<http://tinyurl.com/py3zodb>

C

- Effizienz: nach Möglichkeit Py_UNICODE-Operationen durchführen
 - Py_UNICODE_strncmp etc.
- Beispiel: unghost_getattr

```
static Py_UNICODE class__[] = {'c','l','a','s','s','_','_',0};
```

```
static Py_UNICODE el__[] = {'e','l','_','_',0};
```

```
static Py_UNICODE ict__[] = {'i','c','t','_','_',0};
```

```
static Py_UNICODE of__[] = {'o','f','_','_',0};
```

```
static Py_UNICODE setstate__[] = {'s','e','t','s','t','a','t','e','_','_',0};
```

<http://tinyurl.com/py3zodb>

Probleme: pickle

pickle

- Objektserialisierungsbibliothek
- Kern von ZODB
- Performante Version: cPickle (Jim Fulton)
- Zahlreiche ZODB-Features (z.B. Referenzen auf "persistente" externe Objekte)
- Verschiedene Formate (Protokolle) (0-3)
- 3.x: nur noch Modul pickle; benutzt automatisch `_pickle`

<http://tinyurl.com/py3zodb>

inst_persistent_id

- cPickle-Feature: Manche (Instanz-)Objekte werden nicht selbst serialisiert, sondern durch eine externe ID
- verwendet in ZODB zur Speicherung von Objekten, die ein Attribut `_p_oid` haben, sowie Ghosts
- 3.x: old-style classes nicht mehr vorhanden
 - also auch kein `inst_persistent_id`
 - ersetzt durch `Pickler.persistent_id`

<http://tinyurl.com/py3zodb>

find_global

- Nicht-serialisierbare Objekte werden u.U. mit ihrem Namen serialisiert (Funktionen, Klassen)
- Unpickler muss Namen wieder auflösen
- Standard-Verhalten: `__import__(mod);mod.name`
- 2.x: Hook-Funktion `find_global`
 - ZODB: `DB.classFactory`
- 3.x: Hook heißt `find_class` und lässt sich nicht zuweisen, sondern nur in Ableitung überschreiben
 - Lösung: Unpickler-Ableitung

<http://tinyurl.com/py3zodb>

Protokollversionen

- Pickle speichert Daten im Typ-Wert-Format ab (TV)
- Protokoll 0 (ASCII): jeder Wert ist eine Zeile
- Protokoll 1 (binary): Zahlen, Strings binär
- Protokoll 2: Protokollversion in den Daten, kompaktere Kodierung (z.B. `__newobj__`), Protokoll erweiterbar durch extension codes, ...
- Protokoll 3: Unterstützung für bytes

<http://tinyurl.com/py3zodb>

ZODB-Pickle-Protokoll

- aktuell: Protokoll 1
- Problem: 3.x pickle serialisiert `b"Hallo"` als `builtins.bytes(72, 97, 108, 108, 111)`
 - ineffizient
 - inkompatibel mit 2.x, $x < 6$
 - 3.1.x bug: pickle kann Daten nicht wieder einlesen?
- Lösung: Protokoll 3

<http://tinyurl.com/py3zodb>

ZEO

- nicht getestet
- zrpc verwendet weitere undokumentierte cPickle-Features:
 - `cPickle.Pickler(1)` funktioniert (auch ohne File!)

<http://tinyurl.com/py3zodb>

- The documented signature is `Pickler(file, protocol=0)`, but this accepts `Pickler()` and `Pickler(integer)` too. The meaning then is clear as mud, undocumented, and not supported by `pickle.py`. I'm told Zope uses this, but I haven't traced into this code far enough to figure out what it means

Tim Peters, 2003, in `cPickle.c`

ZEO

- nicht getestet
- zrpc verwendet weitere undokumentierte cPickle-Features:
 - `cPickle.Pickler(1)` funktioniert (auch ohne File!)
 - Ergebnis zugänglich über undokumentierte Methode `getvalue()` oder undokumentierten Extra-Parameter für `.dump`:
`pickler.dump(data, 1)`
 - Zweck: Vermeiden von Datenkopien?

<http://tinyurl.com/py3zodb>

Sonstige Probleme

int vs. long

- C: ersetze PyInt_* durch PyLong_*
- 2.x-Kompatibilität noch nicht hergestellt
- z.B. Einführung von ZInteger_FromLong

<http://tinyurl.com/py3zodb>

Modulinitialisierung

- Init-Funktion heißt nicht mehr `init<module>`, sondern `PyInit_<module>`
- `Py_InitModule[4]` gibt's nicht mehr; `PyModule_Create` erwartet Strukturzeiger
- 2.x-Kompatibilität: gemeinsame init-Funktion, per `ifdef` von `init<module>` oder `PyInit_Module` gerufen wird.

<http://tinyurl.com/py3zodb>

Rich Comparisons

- `cmp` nicht mehr unterstützt
- `TimeStamp`-Typ ist aber geordnet
- Lösung: rich comparisons implementieren
(`Py_LT`, `Py_GT`, ...)

<http://tinyurl.com/py3zodb>

2to3

- 3.x: Imports sind absolut, wenn sie nicht durch "." als relativ deklariert werden
- 2to3 überprüft, ob import relativ gemeint ist, indem nach dem importierten Modul gesucht wird
- ZODB: `from cPickleCache import PickleCache`
 - `cPickleCache.so` ist aber noch nicht vorhanden
 - fixer belässt Code, import scheitert
- Lösung:
`from persistent.cPickleCache import PickleCache`

<http://tinyurl.com/py3zodb>

Tests

Testrunner

- Wie läßt man die ZODB-Testsuite laufen?
- Keine Ahnung - README.txt redet von buildout
- `setup.py test` funktioniert nicht: verwendet ungefixte Quellen
- Testerfolg: `python -m ZODB.tests.testZODB`

<http://tinyurl.com/py3zodb>

Zusammenfassung

Status

- Prototypischer ZODB-Port
- verschiedene ZODB-Tutorials ausführbar
- bisher keine 2.x-Kompatibilität für C-Module
 - sollte aber realisierbar sein
- Tests nicht ausführbar

<http://tinyurl.com/py3zodb>

Ausblick

- Integration in Originalquellen
- setuptools: PJE gegenwärtig inaktiv; vielleicht bietet distribute (Tarek Ziadé) eine Lösung
- zope-Pakete: Haltung der Maintainer bisher unbekannt
- Hilfe ist erwünscht!

<http://tinyurl.com/py3zodb>