TECHNISCHE UNIVERSITÄT DRESDEN

Medienzentrum – Abteilung MIT

ZODB Einführung

Dresden, 13.01.2011 Dipl. inf. Ingo Keller

TECHNISCHE UNIVERSITÄT DRESDEN

Übersicht

Motivation

Python Persistenz

ZODB

Beispiel

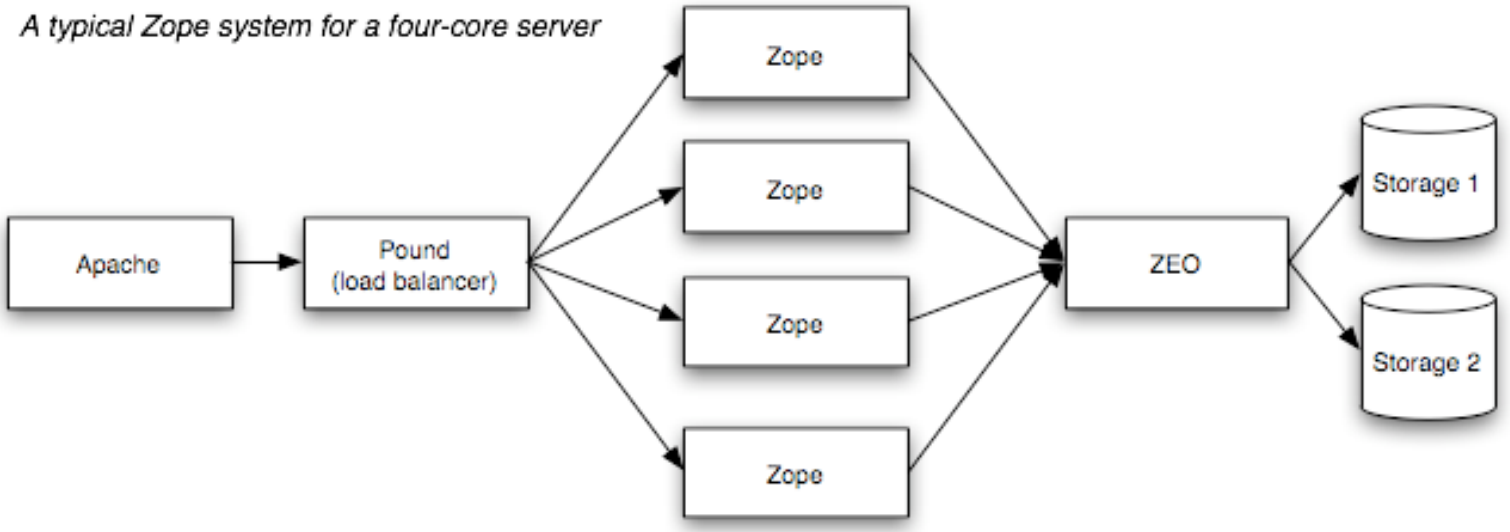
Warum Objektorientierte Datenbanken?

MOTIVATION

Einordnung

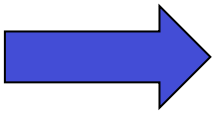
- Datenbanktypen
  - Relational
  - Hierarchisch
  - Objekt-Orientiert

- Multi-Tier Architekturen

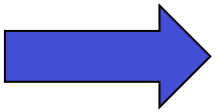


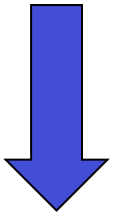
Probleme Relationaler Datenbanken

ID	Name	Vorname	Telefon	EMail	...
1	Mustermann	Max	+41(0)555555	mm@blah.info	...
2	Vorwerk	Frieda		fv@blub.info	...
3	Schildkröte	Paul	0351/232342		...
...	...	...	...	...	...

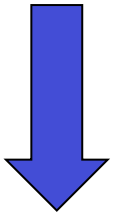


schwierig erweiterbar





einfach erweiterbar



Probleme Relationaler Datenbanken

- Keine Speicherung mehrer Werte in einem Feld
  - Ausnahme spezielle Datentypen
  - > Datenbankabhängig
- Keine Erweiterbarkeit von individuellen Datensätzen
  - Lösung über weitere Tabellen via Fremdschlüssel
  - > Aufwendigeres Verknüpfen über JOINs
- Sehr aufwändig für komplexe Daten
- Kein implizites Konzept von Objektorientierung und Vererbung
  - Lösung über Object-Relational Mapper (z.B. SQLAlchemy)
  - > Extra Entwicklungs- und Laufzeitaufwand

Speichern, aber wie?

PYTHON PERSISTENZ

Pickling

- Modul cPickel
  - Integraler Bestandteil von Python
- Betriebssystem unabhängig
- Serialisierung von Objekten in "ASCII"-Strings
  - kostet Speicherplatz
  - bringt Kodierungsunabhängigkeit

TECHNISCHE UNIVERSITÄT DRESDEN

Pickling Beispiel

```
import cPickle
d = [1,2,3,4]

# Schreiben eines Dumps
file = open('test.pic', 'w')
cPickle.dump(d, file)
file.close()

# Lesen eines Dumps
file = open('test.pic', 'r')
a = cPickle.load(file)
file.close()

# Daten sind wieder da
print a
[1,2,3,4]
```

test.pic

- `cat test.pic`

```
(lp1
I1
aI2
aI3
aI4
a.
```

Und wieso jetzt Datenbank?

ZODB

Installation

- Eigenständiges Python Produkt
- Voraussetzung:
  - Python + setuptools
- Installation:

```
easy_install ZODB3
```

- Nutzung:

```
>>> import ZODB
```

Zope Object Database

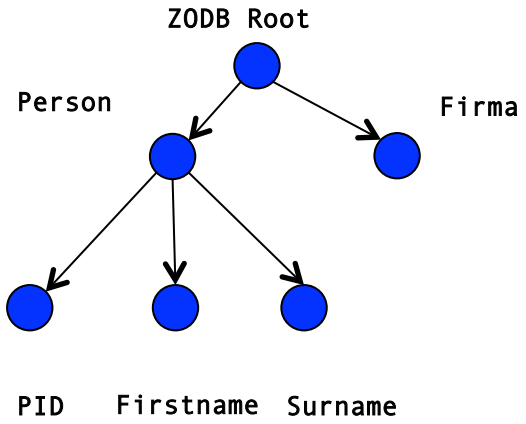
- Transaktionsorientierte Datenbank
  - "two-phase commit" verfügbar
- ACID-fähig
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Undo / History / Timeshift
- Skalierbar (Transaktionsrate, Objektmenge)
- Unterstützt Binary Large Objects (BLOBs)

Zope Object Database

- Ist nicht SQL-fähig
  - NoSQL-Datenbank
- Keine Unterstützung von Replikation
- Langsamer bei tabellenstrukturierten Daten
- Ist nicht Zope
  - wird aber von Zope genutzt

ZODB Kernideen

- Architektur
  - Storage, Datenbank, Wurzelobjekt ("root")
- Transaktionen um Updates zu kontrollieren
  - Sicheres Sharen zwischen Clients
- Persistenz über Erreichbarkeit (Hierarchische DB)
  - Alle erreichbaren Objekte werden gespeichert



Storagetypen

- FileStorage
  - einfach eine große Datei
  - Achtung: maximale Dateigröße des Dateisystems bedenken!
- MappingStorage
  - Speichert Daten in einer In-Memory Datenbank
- DemoStorage
  - Wrapper um einen beliebigen Storage
  - Transaktionen werden in einen separaten Storage geschrieben
  - gewrappter Storage bleibt erhalten
- und noch viele mehr

Basisfunktionalität

```
# Notwendige Imports
from   ZODB       import FileStorage, DB
from   persistent import Persistent
import transaction

# Verbindung zu einer Datenbank
storage = FileStorage.FileStorage('/tmp/Data.fs')
db      = DB(storage)
conn    = db.open()

# Root der Datenbank anfragen
dbroot  = conn.root()

# Festschreiben oder abrechen einer Transaktion
transaction.commit()
transaction.abort()
```

Persistenzklasse

- ZODB kann alle pickelbaren Objekt speichern
  - nicht pickelbar: z.B. Sockets, Device Files

- Klasse `Persistent` als Baseclass für alle speicherbaren Klassen
  - Kann als Mixin genutzt werden
  - bietet alle Persistenzhooks
  - transparente Persistenz

- _p_ - Attribute managen den Persistenzzustand

_p_changed	– Verändert nach letztem Commit ?
_p_mtime	– Modifikationszeit
_p_oid	– OID des Objekts
_p_estimated_size	– Erwartete Recordgröße

Transaktionen

- transaction.begin()
  - implizit über die Connections gegeben
- transaction.commit()
  - nur persistente Objekte automatisch betroffen!
- transaction.abort()
  - Transaktionsabbruch gilt nur für Datenbankobjekte
- transaction.savepoint()
  - neuere Savepoints werden ungültig beim zurückrollen zu älteren
- transaction.doom()

Minimal Beispiel

```
from persistent import Persistent
from ZODB       import FileStorage, DB
import transaction

class Counter(Persistent):
        _value = 0
        def inc(self):
                self._value += 1

def main():
        fs      = DB(FileStorage.FileStorage("Data.fs"))
        conn    = db.open()
        root    = conn.root()
        obj     = root["myCounter"] = Counter()
        transaction.commit()
        obj.inc()
        transaction.commit()


main()
```

OIDs & Records

- OIDs sind 8 Byte Binärstrings die eine 64bit Zahl kodieren
  - es gilt immer: `conn.root()._p_oid == 0`
- Record entspricht einem Einzelobjektpickle
  - Zugreifbar per OID
  - > `db._storage.load(oid)`
- Persistenzmechanismus verknüpft Objekte automatisch
  - native Datentypen werden im Objekt gespeichert
  - > verwenden von `PersistentList`, `PersistentDict` empfohlen
- Datenbankänderung über Hinzufügen neuer Records
  - OIDs wachsen monoton

TIDs & Undo

- TIDs – Transaktionsidentifier
  - sind im wesentlichen Zeitstempel & wachsen monoton
- OIDs repräsentieren den Objektgraphen
  - TIDs repräsentieren die zeitlichen Operationen auf dem Graph
- Undo
  - Zurücksetzen von TIDs

```
>>> db  = ZODB.DB('Data.fs')
>>> tid = db.lastTransaction()
>>> db.undo(tid)
```

History / Time Travel

- History
  - TID-Graph eines Objekts
- Time Travel
  - Temporäres zurücksetzen des Zeigers auf den aktuellsten TID

TECHNISCHE UNIVERSITÄT DRESDEN

Packen & Backup

- Storages wachsen immer nur
  - Löschen entspricht Anfügen aktualisierter Daten mit neuem TID
- Packen entfernt alle "veralteten" Transaktionen
  > regelmässiges Packen notwendig
- Hot Snapshots möglich
  - simples kopieren der Data.fs
  > bei gleichzeitigem Schreiben besteht Gefahr von Datenkorruption
  > aber nur die letzten Transaktionen! Konsistenz bleibt gewahrt
- Backuptool: repozo

ZODB Tools

- Analyseskript:

```
>>> from ZODB.scripts.analyze import analyze, report
>>> ...
>>> print report(analyze(storage))
```

```
Processed 4084 records in 11 transactions
Average record size is  267.07 bytes
Average transaction size is 99154.27 bytes
Types used:
Class Name                                Count   TBytes     Pct  AvgSize
--------------------------------------- ------- ---------  ----- -------
AccessControl.User.User                       1       141   0.0%  141.00
AccessControl.User.UserFolder                 1       103   0.0%  103.00
App.ApplicationManager.ApplicationManager     1       107   0.0%  107.00
```

ZODB Tools

- checkbtrees
  - rekursives checken aller BTrees
- fsoids
  - OID Tracer, liefer alle Informationen zu gegebenen OIDs
- fsrefs
  - checken von "Hängenden" Referenzen
- fstail
  - Dumped die letzten Transaktionen (wie tail)
- fstest
  - Konsistenzchecker

Was kann man nun damit machen?

BEISPIEL

TECHNISCHE UNIVERSITÄT DRESDEN

Quellen

- "Das Python Praxisbuch", F. Hajji, Addison-Wesley, 2008
- "GoTo Python", D. Harms, Addison-Wesley, 2003
- "Persistente Python-Objekte mit der ZODB", Chr. Theune, Workshop DZUG, 2009
- http://www.zodb.org

Storage Klasse

```
from persistent import Persistent

class Person (Persistent):

        def __init__(self, pid, firstname, surname):
                self.pid       = pid
                self.firstname = firstname
                self.surname   = surname

        def __str__(self):
                return 'Person(%s, %s, %s) % (self.pid, self.firstname,
                        self.surname)
```

Abgeleitete Storage Klasse

```
class PersonNG (Persistent):

        def __init__(self, pid, firstname, surname, email, website):
                Person.__init__(pid, firstname, surname)
                self.email     = email
                self.website   = website

        def __str__(self):
                return 'PersonNG(%s, %s, %s, %s, %s) % (
                        self.pid,
                        self.firstname,
                        self.surname,
                        self.email,
                        self.url)
```