Practical Google App Engine Applications in Python

上官林傑 (ericstk)
COSCUP 2009

http://tinyurl.com/coscup-appengine
Outline

- Effective Datastore API
- Data Manipulation Efficiency
- Effective Memcache
- Zip Import & Zip Serve
- Conclusion
Quota Limit on App Engine

from: http://www.flickr.com/photos/kevin814/3587610731/
What's Datastore

- Datastore is a kind of **key-value database** built on GFS.
  - scalable
  - Kind-based data entities. (not table-based)
  - add/remove properties dynamically
Avoid Heavily-Indexing

- Datastore will create index on each property.
  - If there're many properties in your data, **indexing will downgrade performance**.
  - If a property is not used for filtering nor ordering, add `indexed=False` to the data model declaration.

```python
class Foo(db.Model):
    name = db.StringProperty(required=True)
    bar = db.StringProperty(indexed=False)
```
Minimize Datastore API Calls

- **CRUD data entities by keys:**
  - **Ineffective Way:**
    ```python
    keys = [key1, key2, key3, ..., keyN]
    products = []
    for key in keys:
        products.append(db.get(key))
    ...
    ```
  - **Effective Way:**
    ```python
    keys = [key1, key2, key3, ..., keyN]
    products = db.get(keys)
    
    Same as `db.put()`, `db.delete()`.
    ```
Re-bind GqlQuery Object

● Use prepared GqlQuery data:
  o Ineffective way:
    conds = [['abc', 'def'], ['123', '456'], ...]
    for cond in conds:
      query = db.GqlQuery('SELECT * FROM Foo WHERE first = :first, second = :second', first=cond[0], second=cond[1])
  ....
  o Effective way:
    conds = [['abc', 'def'], ['123', '456'], ...]
    prepared_query = db.GqlQuery('SELECT * FROM Foo WHERE first = :first, second = :second')
    for cond in conds:
      query = prepared_query.bind(first=cond[0], second=cond[1])
  ....
Avoid Disjunctions

- **IN** or **!=** operator generates more queries.
  - `SELECT * FROM Foo WHERE a IN ('x', 'y') and b != 3`
    - splits into 4 queries
      - `SELECT * FROM Foo WHERE a == 'x'`
      - `SELECT * FROM Foo WHERE a == 'y'`
      - `SELECT * FROM Foo WHERE b < 3`
      - `SELECT * FROM Foo WHERE b > 3`

- Fetches all data and filters them manually.
How to Fetch More Than 1000 Results

- Datastore API fetches no more than 1000 results once a call
- Fetches more than 1000 results (SLOW, may cause TLE)

```python
data = Foo.gql('ORDER BY __key__').fetch(1000)
last_key = data[-1].key()
results = data

while len(data) == 1000:
    data = Foo.gql('WHERE __key__ > :1 ORDER BY __key__', last_key).fetch(1000)
    last_key = data[-1].key()
    results.extend(data)
```
Put Data into Entity Group
Put data into an entity group:

```python
forum = Forum.get_by_key_name('HotForum')

parent = forum

for data in data_to_put:
    topic = Topic(key_name='Topic1', parent=forum).put()
```

Load data from an entity group:

```python
parent = db.Key.from_path('Forum', 'HotForum')

for data in data_to_load:
    topic = Topic.get_by_key_name('Topic1', parent=parent)
```
Sharding Data

- Write data in parallel
  - avoiding write contention
- Sharding data with key_name:

```python
class Counter(db.Model):
    name = db.StringProperty()
    count = db.IntegerProperty()

...

def incr_counter(counter_name):
    shard = random.randint(0, NUM_SHARDS - 1)
    counter = Counter.get_or_insert(shard, name=counter_name)
    counter.count += 1
    counter.put()
```
Effective Caching

- Caching page content
  - Without caching
    ```python
    ....
    self.response.out.write(
        template.render('index.html', {})
    )
    ....
    ```
  - With Caching
    ```python
    page_content = memcache.get('index_page_content')
    if page_content is None:
        page_content = template.render('index.html', {})  # Change to:
    self.response.out.write(page_content)
    ```
Effective Caching (cont.)

- Caching frequently fetched entities
  - Without caching
    ```python
    ....
    products = Product.gql('WHERE price < 100').fetch(1000)
    from django.utils import simplejson
    self.response.out.write(simplejson.dumps(products))
    ```
  - With caching
    ```python
    ...
    products = memcache.get('products_lt_100')
    if products is None:
        products = Product.gql('WHERE price < 100').fetch(1000)
    from django.utils import simplejson
    self.response.out.write(simplejson.dumps(products))
    ```
Zipimport & ZipServe

- **ZipImport:**
  Zip your library and then import modules within it.

- **ZipServe:**
  Zip your static/asset files, then serve them with `zipserve`.

- **WHY?**
  You can **ONLY** put 1000 files in your application.
For example, you want to use the **Google Data client library** in your application.

- You have to put `gdata/` and `atom/` packages into your application directory.
- With `zipimport`, you can zip them:
  ```
  application/
  app.yaml
  ....
  atom.zip
  gdata.zip
  ....
  ```
Zipimport (cont.)

- import gdata modules in your script:
  ...
  import sys

  
  sys.path.append('atom.zip')
  sys.path.append('gdata.zip')
  ....
  from gdata.doc import import service
For example, you want to use TinyMCE library
  o You have to put TinyMCE library into your directory. However, it contains lots of files.
  o With zipserve, you can zip the library, and configure the app. yaml:

    ... 
    - url: /tinymce/.*
      script: $PYTHON_LIB/google/appengine/ext/zipserve

  o The filename **MUST** be the same as the directory name. In this sample, the TinyMCE library should be zipped into tinymce. zip.
Conclusion - How to Write Better GAE Apps?

- Read the Articles on Google App Engine site.
- Trace the source from SDK
  - Maybe you will find the undocumented API.
- Develop apps!
台北, Taipei

http://taipei-gtug.org/