hypatia Documentation

Release 0.4.dev0

Agendaless Consulting

April 04, 2016

Contents

`hypatia` is a Python indexing and searching framework. It works on Python 2.7+ and Python 3.3+.

CHAPTER 1

Narrative documentation

Narrative documentation explaining how to use `hypatia`.

Warning: This is under construction during the alpha phase of Hypatia.

1.1 Genealogy of `hypatia`

Hypatia is derived from *zope.index* and *repoze.catalog*.

Hypatia depends heavily on the Zope Object Database (ZODB). Because ZODB is less a database and more a persistent object store (it doesn't possess a query language; Python *is* its query language), it has been necessary to create indexing and searching facilities for data stored in ZODB.

The first iteration of searching and indexing for ZODB-based applications (at least post-Principia, which had something named Tabula, which I never actually used) was the ZCatalog. The ZCatalog was entirely tied to Zope2, and still remains in heavy use today within Zope 2 applications such as Plone.

The second iteration was `zope.app.catalog`, which was a ZCatalog do-over for Zope 3 applications.

Neither of these searching and indexing packages are particularly easy to use outside of a Zope application. Each makes various assumptions about the content objects that need indexing or the environment that aren't appropriate for arbitrary applications. For instance, ZCatalog wants objects you want to catalog to have a `getPhysicalPath` method which returns a "path". An instance of `zope.app.catalog` makes the assumption that that it's located within a Zope 3 "site" object within a ZODB, and assumes that you want query result sets to be sets of Python references to the original object you indexed. In other words, these packages assume too much to be maximally useful outside the context in which they were developed. Repoze is a project which has as a stated goal making it easier for non-Zope Python developers to use Zope technologies outside Zope, so this seemed like a natural thing to do under the Repoze flag.

Hypatia is a reboot of both zope.index and repoze.catalog with no backwards compatibility constraints.

1.2 Installing `hypatia`

1.2.1 How To Install

You will need Python version 2.6 or better to run `hypatia`. Development of `hypatia` is done primarily under Python 2.7, so that version is recommended. It does *not* run under Python 3.X.

Warning: To succesfully install `hypatia`, you will need an environment capable of compiling Python C code. See the documentation about installing, e.g. `gcc` and `python-devel` for your system. You will also need *setuptools* installed within your Python system in order to run the `easy_install` command.

It is advisable to install `hypatia` into a *virtualenv* in order to obtain isolation from any "system" packages you've got installed in your Python version (and likewise, to prevent `hypatia` from globally installing versions of packages that are not compatible with your system Python).

After you've got the requisite dependencies installed, you may install `hypatia` into your Python environment using the following command:

```
$ easy_install hypatia
```

1.2.2 What Gets Installed

When you `easy_install hypatia` and ZODB are installed.

1.3 Glossary

Setuptools Setuptools builds on Python's `distutils` to provide easier building, distribution, and installation of packages.

Interface An attribute of a model object that determines its type. It is an instance of a `zope.interface` Interface class.

Zope The Z Object Publishing Framework. The granddaddy of Python web frameworks.

ZODB The Zope Object Database which is a persistent object store for Python.

Field index A type of index that is optimized to index single simple tokenized values. When a field index is searched, it can be searched for one or more values, and it will return a result set that includes these values exacty.

Text index A type of index which indexes a value in such a way that parts of it can be searched in a non-exact manner. When a text index is searched, it returns results for values that match based on various properties of the text indexed, such as omitting "stopwords" the text might have.

Facet index A type of index which can be used for faceted search.

Path index A type of index that keeps track of documents within a graph; documents can be searched for by their position in the graph.

zope.index One package that hypatia was forked from.

repoze.catalog Another package that hypatia was forked from.

Virtualenv An isolated Python environment. Allows you to control which packages are used on a particular project by cloning your main Python. virtualenv was created by Ian Bicking.

CQE A string representing a Python-like domain-specific-language expression which is used to generate a query object.

Query Object An object used as an argument to the `hypatia.CatalogQuery.__call__()` method's `queryobject` parameter.

1.4 `hypatia` Change History

CHAPTER 2

API documentation

API documentation for `hypatia`.

2.1 hypatia.catalog

class `hypatia.catalog.`**Catalog**(*family=None*)

__setitem__(*name*, *index*)

__getitem__(*key*)
Retrieve an index.

get(*key*, *failobj=None*)
Retrieve an index or return failobj.

reset()
Clear all indexes in this catalog.

index_doc(*docid*, *obj*)
Register the document represented by `obj` in indexes of this catalog using docid `docid`.

unindex_doc(*docid*)
Unregister the document id from indexes of this catalog.

reindex_doc(*docid*, *obj*)
Reindex the document referenced by docid using the object passed in as `obj` (typically just does the equivalent of `unindex_doc`, then `index_doc`, but specialized indexes can override the method that this API calls to do less work.

index_doc(*docid*, *obj*)
Register the document represented by `obj` in indexes of this catalog using docid `docid`.

reindex_doc(*docid*, *obj*)
Reindex the document referenced by docid using the object passed in as `obj` (typically just does the equivalent of `unindex_doc`, then `index_doc`, but specialized indexes can override the method that this API calls to do less work.

reset()
Clear all indexes in this catalog.

unindex_doc(*docid*)
Unregister the document id from indexes of this catalog.

class hypatia.catalog.**CatalogQuery**(*catalog*, *family=None*)

Legacy query API for non-index-based queries; might be useful if/when an index-based query doesn't work properly, or a particular contstraint can't be spelled with one.

__call__(*queryobject*, *sort_index=None*, *limit=None*, *sort_type=None*, *reverse=False*, *names=None*)

Use the arguments to perform a query. Return a tuple of (num, resultseq).

sort(*docidset*, *sort_index*, *limit=None*, *sort_type=None*, *reverse=False*)

Return `(num, sorted-resultseq)` for the concrete docidset.

query(*queryobject*, *sort_index=None*, *limit=None*, *sort_type=None*, *reverse=False*, *names=None*)

Use the arguments to perform a query. Return a tuple of (num, resultseq).

search(***query*)

Use the query terms to perform a query. Return a tuple of (num, resultseq) based on the merging of results from individual indexes.

Note: This method is deprecated. Use *hypatia.catalog.CatalogQuery.__call__()* instead.

---

sort(*docidset*, *sort_index*, *limit=None*, *sort_type=None*, *reverse=False*)

Return `(num, sorted-resultseq)` for the concrete docidset.

2.2 hypatia.query

2.2.1 Comparators

class hypatia.query.**Contains**(*index*, *value*)

Contains query.

CQE equivalent: 'foo' in index

class hypatia.query.**Eq**(*index*, *value*)

Equals query.

CQE equivalent: index == 'foo'

class hypatia.query.**NotEq**(*index*, *value*)

Not equal query.

CQE eqivalent: index != 'foo'

class hypatia.query.**Gt**(*index*, *value*)

Greater than query.

CQE equivalent: index > 'foo'

class hypatia.query.**Lt**(*index*, *value*)

Less than query.

CQE equivalent: index < 'foo'

class hypatia.query.**Ge**(*index*, *value*)

Greater (or equal) query.

CQE equivalent: index >= 'foo'

class hypatia.query.**Le**(*index*, *value*)

Less (or equal) query.

CQE equivalent: index <= 'foo

class hypatia.query.**Contains**(*index*, *value*)
Contains query.

CQE equivalent: 'foo' in index

class hypatia.query.**NotContains**(*index*, *value*)
CQE equivalent: 'foo' not in index

class hypatia.query.**Any**(*index*, *value*)
Any of query.

CQE equivalent: index in any(['foo', 'bar'])

class hypatia.query.**NotAny**(*index*, *value*)
Not any of query (ie, None of query)

CQE equivalent: index not in any(['foo', 'bar'])

class hypatia.query.**All**(*index*, *value*)
All query.

CQE equivalent: index in all(['foo', 'bar'])

class hypatia.query.**NotAll**(*index*, *value*)
NotAll query.

CQE equivalent: index not in all(['foo', 'bar'])

class hypatia.query.**InRange**(*index*, *start*, *end*, *start_exclusive=False*, *end_exclusive=False*)
Index value falls within a range.

CQE eqivalent: lower < index < upper lower <= index <= upper

class hypatia.query.**NotInRange**(*index*, *start*, *end*, *start_exclusive=False*, *end_exclusive=False*)
Index value falls outside a range.

CQE eqivalent: not(lower < index < upper) not(lower <= index <= upper)

2.2.2 Boolean Operators

class hypatia.query.**Or**(**queries*)
Boolean Or of multiple queries.

class hypatia.query.**And**(**queries*)
Boolean And of multiple queries.

class hypatia.query.**Not**(*query*)
Negation of a query.

2.2.3 Other Helpers

class hypatia.query.**Name**(*name*)
A variable name in an expression, evaluated at query time. Can be used to defer evaluation of variables used inside of expressions until query time.

Example:

```
    from hypatia.query import Eq
    from hypatia.query import Name

    # Define query at module scope
    find_cats = Eq('color', Name('color')) & Eq('sex', Name('sex'))

    # Use query in a search function, evaluating color and sex at the
    # time of the query
    def search_cats(catalog, resolver, color='tabby', sex='female'):
        # Let resolver be some function which can retrieve a cat object
        # from your application given a docid.
        params = dict(color=color, sex=sex)
        count, docids = catalog.query(find_cats, params)
        for docid in docids:
            yield resolver(docid)
```

hypatia.query.**parse_query**(*expr*, *catalog*, *optimize_query=True*)
Parses the given expression string and returns a query object. Requires Python >= 2.6.

2.3 hypatia.util

class hypatia.util.**ResultSet**(*ids*, *numids*, *resolver*, *sort_type=None*)
Implements hypatia.interfaces.IResultSet

intersect(*docids*)
Intersect this resultset with a sequence of docids or another resultset. Returns a new ResultSet.

2.4 hypatia.exc

class hypatia.exc.**BadResults**(*resultset*)
Superclass of *hypatia.exc.MultipleResults* and *hypatia.exc.NoResults*. Has an attribute named resultset which is the resultset related to the error.

class hypatia.exc.**MultipleResults**(*resultset*)
Raised when a method that was expected to return a single result returns multiple results.

class hypatia.exc.**NoResults**(*resultset*)
Raised when a method that was expected to return at least one result returns zero results.

class hypatia.exc.**Unsortable**(*docids*)
Raised when a method which was expected to sort a set of provided document identifiers cannot sort one or more of those identifiers. An attribute named docids is a sequence containing the identifiers.

2.5 hypatia.field

Field index

class hypatia.field.**FieldIndex**(*discriminator*, *family=None*)
Field indexing.

Query types supported:

- Eq

- NotEq
- Gt
- Ge
- Lt
- Le
- In
- NotIn
- Any
- NotAny
- InRange
- NotInRange

index_doc(*docid*, *value*)
See interface IIndexInjection

reindex_doc(*docid*, *value*)
See interface IIndexInjection

reset()
Initialize forward and reverse mappings.

unindex_doc(*docid*)
See interface IIndexInjection.

unique_values()
Return the unique values in the index for all docids as an iterable

word_count()
See interface IIndexStatistics

2.6 hypatia.keyword

class hypatia.keyword.**KeywordIndex**(*discriminator*, *family=None*)
Keyword index.

Query types supported:

- Eq
- NotEq
- In
- NotIn
- Any
- NotAny
- All
- NotAll

normalize(*seq*)
Perform normalization on sequence of keywords.

Return normalized sequence. This method may be overriden by subclasses.

optimize()
Optimize the index. Call this after changing tree_threshold.

This converts internal data structures between Sets and TreeSets based on tree_threshold.

reset()
Initialize forward and reverse mappings.

search(*query*, *operator='and'*)
Execute a search given by 'query'.

unique_values()
Return the unique values in the index for all docids as an iterable

word_count()
Return the number of indexed words

2.7 hypatia.text

Text index.

class hypatia.text.**TextIndex**(*discriminator*, *lexicon=None*, *index=None*, *family=None*)

check_query(*querytext*)
Returns True if the querytext can be turned into a parse tree, returns False if otherwise.

sort(*result*, *reverse=False*, *limit=None*, *sort_type=None*, *raise_unsortable=True*)
Sort by text relevance.

This only works if the query includes at least one text query, leading to a weighted result. This method raises TypeError if the result is not weighted.

A weighted result is a dictionary-ish object that has docids as keys and floating point weights as values. This method sorts the dictionary by weight and returns the sorted docids as a list.

word_count()
Return the number of words in the index.

2.8 hypatia.facet

class hypatia.facet.**FacetIndex**(*discriminator*, *facets*, *family=None*)
Facet index.

Query types supported:

- Eq
- NotEq
- In
- NotIn
- Any

- NotAny
- All
- NotAll

counts (*docids*, *omit_facets=()*)
Given a set of docids (usually returned from query), provide count information for further facet narrowing. Optionally omit count information for facets and their ancestors that are in 'omit_facets' (a sequence of facets)

index_doc (*docid*, *obj*)
Pass in an integer document id and an object supporting a sequence of facet specifiers ala ['style:gucci:handbag'] via the discriminator

2.9 hypatia.interfaces

hypatia.interfaces.**STABLE**
Used as an argument to the `sort_type` parameter of IIndexSort.sort.

hypatia.interfaces.**OPTIMAL**
Used as an argument to the `sort_type` parameter of IIndexSort.sort.

CHAPTER 3

Source Code and Issue Tracking

Source code is available from https://github.com/Pylons/hypatia

File bugs via https://github.com/Pylons/hypatia/issues

3.1 Indices and tables

- genindex
- modindex
- search
- *Glossary*

h

Symbols

A

B

C

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z