
parasolr Documentation

Release 0.10.0-dev

Center for Digital Humanities, Princeton University

Jun 23, 2023

CONTENTS:

1	Code Documentation	1
2	Getting Started	23
3	CHANGELOG	25
4	Installation	31
5	Development instructions	33
6	Unit testing	35
7	License	37
8	Indices and tables	39
	Python Module Index	41
	Index	43

CODE DOCUMENTATION

1.1 Solr Client

1.1.1 Base and Exceptions

class `parasolr.solr.base.ClientBase(session=None)`

Base object with common communication methods for talking to Solr API.

Parameters

session (`Optional[Session]`) – A python-requests `requests.Session`.

build_url (`solr_url`, `collection`, `handler`)

Return a url to a handler based on core and base url.

Parameters

- **solr_url** (`str`) – Base url for Solr.
- **collection** (`str`) – Collection or core name.
- **handler** (`str`) – Handler URL for construction.

Return type

`str`

Returns

A full-qualified URL.

make_request (`meth`, `url`, `headers=None`, `params=None`, `data=None`, `wrap=True`, `allowed_responses=None`, `**kwargs`)

Make an HTTP request to Solr. May optionally specify a list of allowed HTTP status codes for this request. Responses will be logged as errors if they are not in the list, but only responses with 200 OK status will be loaded as JSON.

Parameters

- **meth** (`str`) – HTTP method to use.
- **url** (`str`) – URL to make request to.
- **headers** (`Optional[dict]`) – HTTP headers.
- **params** (`Optional[dict]`) – Params to use as form-fields or query-string params.
- **data** (`Optional[dict]`) – Data for a POST request.
- **allowed_responses** (`Optional[list]`) – HTTP status codes that are allowed for this request; if not set, defaults to 200 OK

- ****kwargs** (*Any*) – Any other kwargs for the request.

Return type

Optional[Dict]

exception parasolr.solr.base.CoreExists

Raised when a Solr core exists and it should not.

exception parasolr.solr.base.ImproperConfiguration

Raised when a required setting is not present or is an invalid value.

exception parasolr.solr.base.SolrClientException

Base class for all exceptions in this module

exception parasolr.solr.base.SolrConnectionNotFound

Raised when a 404 is returned from Solr (e.g., attempting to query a non-existent collection on a valid Solr server)

1.1.2 Client and Search API

class parasolr.solr.client.BaseResponse(*response*)

Base Solr response class with fields common to standard and grouped results.

_process_facet_counts(*facet_counts*)

Convert facet_fields and facet_ranges to OrderedDict.

Parameters

facet_counts (Dict) – Solr facet_counts field.

Return type

OrderedDict

Returns

Solr facet_counts field

property expanded

expanded portion of the response, if collapse/expanded results enabled

property highlighting

highlighting portion of the response, if highlighting was requested

property params

parameters sent to solr in the request, as returned in response header

property stats

stats portion of the response, if stats were requested

class parasolr.solr.client.GroupedResponse(*response*)

Query response variant for grouped results.

Parameters

response (Dict) – A Solr query response

property group_field: *str*

group.field as stored in the params. Not yet supporting grouping by query.

property groups: [List](#)

Unlike *QueryResponse.docs*, this returns a list of groups with nested documents.

Returns

`_description_`

Return type

List

class `parasolr.solr.client.ParasolrDict(*args, **kwargs)`

A subclass of `addict.Dict` that can convert itself to a regular dictionary.

as_dict()

Copy attributes from self as a dictionary, and recursively convert instances of *ParasolrDict*.

class `parasolr.solr.client.QueryResponse(response)`

Thin wrapper to give access to Solr select responses.

Parameters

response (*Dict*) – A Solr query response

class `parasolr.solr.client.SolrClient(solr_url, collection, commitWithin=None, session=None)`

Class to aggregate all of the other Solr APIs and settings.

Parameters

- **solr_url** (*str*) – Base url for solr.
- **collection** (*str*) – Name of Solr collection or core.
- **commitWithin** (*Optional[int]*) – Time in ms for soft commits to happen.
- **session** (*Optional[Session]*) – A python-requests *requests.Session*.

collection = ''

core or collection name

commitWithin = 1000

commitWithin time in ms

core_admin_handler = 'admin/cores'

CoreAdmin API handler

query(*wrap=True, **kwargs*)

Perform a query with the specified kwargs.

Parameters

****kwargs** (*Any*) – Any valid Solr search parameters.

Return type

Optional[QueryResponse]

Returns

A search *QueryResponse*.

schema_handler = 'schema'

Schema API handler

select_handler = 'select'

Select handler

1.1.3 Schema API

Module with class and methods for the Solr Schema API.

class `parasolr.solr.schema.Schema(solr_url, collection, handler, session=None)`

Class for managing Solr Schema API

Parameters

- **solr_url** (`str`) – Base url for Solr.
- **collection** (`str`) – Name of the collection or core.
- **handler** (`str`) – Handler name for Solr Schema API.
- **session** (`Optional[Session]`) – A python-requests `requests.Session`.

_post_field(*method*, ***field_kwargs*)

Post a field definition to the schema API.

Parameters

- **method** (`str`) – Solr field method to use.
- ****field_kwargs** (`Any`) – Field arguments to use in definition. Any valid schema definition may be used; if passed as `kwargs`, rather than `dict`, `klass` may be used instead of `class`.

Return type

`None`

add_copy_field(*source*, *dest*, *maxChars=None*)

Add a copy field between two existing fields.

Parameters

- **source** (`str`) – Source Solr field.
- **dest** (`str`) – Destination Solr field.
- **maxChars** (`Optional[int]`) – Maximum characters to copy.

Return type

`None`

add_field(***field_kwargs*)

Add a field with the supplied definition.

Parameters

- ****field_kwargs** (`Any`) – Any valid Solr field definition values.

Return type

`None`

add_field_type(***field_kwargs*)

Add a field type to a Solr collection or core.

Parameters

- ****field_kwargs** (`Any`) – Any valid Solr field definition values.

Return type

`None`

Returns

`None`

delete_copy_field(*source*, *dest*)

Delete a Solr copy field.

Parameters

- **source** (*str*) – Source Solr field.
- **dest** (*str*) – Destination Solr field.

Return type

None

delete_field(*name*)

Delete a field with the supplied name.

Parameters

name (*str*) – Name of field to delete.

Return type

None

delete_field_type(*name*)

Delete a field type from a Solr collection or core.

Parameters

name (*str*) – Name of Solr field type to delete.

Return type

None

get_schema()

Get the full schema for a Solr collection or core.

Return type

Dict

Returns

Schema as returned by Solr.

list_copy_fields(*source_fl=None*, *dest_fl=None*)

Return a list of copy fields from Solr.

Parameters

- **source_fl** (*Optional[list]*) – Source field to filter by.
- **dest_fl** (*Optional[list]*) – Destination field to filter by.

Return type

List[Dict]

Returns

list of copy fields as returned by Solr.

list_field_types(*showDefaults=True*)

List all field types in a Solr collection or core.

Parameters

showDefaults (*bool*) – Show default fields

Return type

List[Dict]

Returns

list of copy fields as returned by Solr.

list_fields(*fields=None, includeDynamic=False, showDefaults=False*)

Get a list of field definitions for a Solr Collection or core.

Parameters

- **fields** (*Optional[list]*) – A list of fields to filter by.
- **includeDynamic** (*bool*) – Include Solr dynamic fields in search.
- **showDefaults** (*bool*) – Show default Solr fields.

Return type

list

Returns

list of fields as returned by Solr.

replace_field(***field_kwargs*)

Replace a field with the supplied definition

Parameters

****field_kwargs** (*Any*) – Any valid Solr field definition values; must be a full redefinition, not a partial update.

Return type

None

replace_field_type(***field_kwargs*)

Replace a field type from a Solr collection or core.

Parameters

****field_kwargs** (*Any*) – Any valid Solr field definition values, but must be a full redefinition, not a partial update.

Return type

None

1.1.4 Update API

class parasolr.solr.update.**Update**(*solr_url, collection, handler, commitWithin, session=None*)

API client for Solr update functionality.

Parameters

- **solr_url** (*str*) – Base url for Solr.
- **handler** (*str*) – Handler for Update API.
- **session** (*Optional[Session]*) – A python-requests *requests.Session*.

_delete(*del_obj*)

Private method to pass a delete object to the update handler.

Parameters

del_obj (*Union[dict, list]*) – Object to be serialized into valid JSON for Solr delete.

Return type

None

delete_by_id(*id_list*)

Delete documents by id field.

Parameters

id_list (*list*) – A list of ids.

Return type

None

delete_by_query(*query*)

Delete documents by an arbitrary search query.

Parameters

query (*str*) – Any valid Solr query.

Return type

None

index(*docs*, *commit=False*, *commitWithin=None*)

Index a document or documents, by default with a soft commit.

Parameters

- **docs** (*list*) – list of *dict* objects to index.
- **commit** (*bool*, *optional*) – Whether or not to make a hard commit to the index.
- **commitWithin** (*int*, *optional*) – Override default commitWithin for soft commits.

Return type

None

1.1.5 Core Admin API

class `parasolr.solr.admin.CoreAdmin`(*solr_url*, *handler*, *session=None*)

API client for Solr core admin.

Parameters

- **solr_url** (*str*) – Base url for Solr.
- **handler** (*str*) – Handler for CoreAdmin API
- **session** (*Optional[Session]*) – A python-requests *requests.Session*

create(*name*, ***kwargs*)

Create a new core and register it.

Parameters

- **name** (*str*) – Name of core to create.
- ****kwargs** (*Any*) – Any valid parameter for core creation in Solr.

Return type

None

ping(*core*)

Ping a core to check status.

Parameters

core (*str*) – Name of core to ping.

Return type

`bool`

Returns

True if core status is OK, otherwise False.

reload(*core*)

Reload a Solr Core.

Parameters

core (`str`) – Name of core to reload.

Return type

`None`

status(*core=None*)

Get the status of all cores or one core.

Parameters

core (`Optional[str]`) – Name of core to get status.

Return type

`Optional[Dict]`

unload(*core*, *kwargs*)**

Unload a core, without defaults to remove data dir or index.

Parameters

- **core** (`str`) – Name of core to unload.
- ****kwargs** (`Any`) – Any valid parameter for core unload in Solr.

Return type

`None`

1.2 Schema

Solr schema configuration and management.

Extend `SolrSchema` for your project and configure the fields, field types, and copy fields you want defined in Solr. Fields should be defined using `SolrField` and field types with `SolrAnalyzer` and `SolrFieldType`. For example:

```
from parasolr import schema

class MySolrSchema(schema.SolrSchema):
    """Project Solr schema configuration"""

    # field declarations
    author = schema.SolrField('text_en')
    author_exact = schema.SolrStringField()
    title = schema.SolrField('text_en')
    title_nostem = schema.SolrStringField()
    subtitle = schema.SolrField('text_en')
    collections = schema.SolrField('text_en', multivalued=True)

    #: copy fields, for facets and variant search options
```

(continues on next page)

(continued from previous page)

```
copy_fields = {
    'author': 'author_exact',
    'collections': 'collections_s',
    'title': ['title_nostem', 'title_s'],
    'subtitle': 'subtitle_s',
}
```

Copy fields should be a dictionary of source and destination fields; both single value and list are supported for destination.

If you want to define a custom field type, you can define an analyzer for use in one or more field type declarations:

```
class UnicodeTextAnalyzer(schema.SolrAnalyzer):
    "Solr text field analyzer with unicode folding. Includes all standard
text field analyzers (stopword filters, lower case, possessive, keyword
marker, porter stemming) and adds ICU folding filter factory.
"

    tokenizer = 'solr.StandardTokenizerFactory'
    filters = [
        {"class": "solr.StopFilterFactory", "ignoreCase": True,
         "words": "lang/stopwords_en.txt"},
        {"class": "solr.LowerCaseFilterFactory"},
        {"class": "solr.EnglishPossessiveFilterFactory"},
        {"class": "solr.KeywordMarkerFilterFactory"},
        {"class": "solr.PorterStemFilterFactory"},
        {"class": "solr.ICUFoldingFilterFactory"},
    ]

class SolrTextField(schema.SolrTypedField):
    field_type = 'text_en'

class MySolrSchema(schema.SolrSchema):
    "Schema configuration with custom field types"

    text_en = schema.SolrFieldType('solr.TextField',
                                    analyzer=UnicodeFoldingTextAnalyzer)

    content = SolrTextField()
```

To update your configured solr core with your schema, run:

```
python manage.py solr_schema
```

This will automatically find your *SolrSchema* subclass and apply changes. See *solr_schema* manage command documentation for more details.

class parasolr.schema.SolrAnalyzer

Class to declare a solr field analyzer with tokenizer and filters, for use with *SolrFieldType*.

filters = None

list of the filters to apply

tokenizer = None

string name of the tokenizer to use

class parasolr.schema.SolrField(*fieldtype, required=False, multivalued=False, default=None, stored=True*)

A descriptor for declaring a solr field on a [SolrSchema](#) instance.

Parameters

- **fieldtype** (*str*) – The type of Solr field.
- **required** (*bool*) – Whether the field is required.
- **multivalued** – Whether the field is multi-valued.

Raises

AttributeError – If `__set__` is called.

class parasolr.schema.SolrFieldType(*field_class, analyzer, **kwargs*)

A descriptor for declaring and configure a solr field type on

Parameters

- **field_class** (*str*) – The class of the SolrField
- **analyzer** (*str*) – The name of the Solr analyzer to use on the field.
- **arguments.** (*Additional field options can be passed as keyword*) –

Raises

AttributeError – If `__set__` is called.

class parasolr.schema.SolrSchema

Solr schema configuration.

classmethod **configure_copy_fields**(*solr*)

Update configured Solr instance schema with copy fields.

Parameters

solr (*SolrClient*) – Configured Solr Schema.

Return type

None

classmethod **configure_fields**(*solr*)

Update the configured Solr instance schema to match the configured fields.

Calls [configure_copy_fields\(\)](#) after new fields have been created and before old fields are removed, since an outdated copy field could prevent removal.

Parameters

solr (*SolrClient*) – A configured Solr instance schem.

Return type

Dict

Returns

`attrdict.AttrDefault` with counts for added, updated, and deleted fields.

classmethod **configure_fieldtypes**(*solr*)

Update the configured Solr instance so the schema includes the configured field types, if any.

Parameters

solr (*SolrClient*) – A configured Solr instance.

Return type

Dict

Returns

addict.Dict with counts for updated and added field types.

copy_fields = {}

dictionary of copy fields to be configured key is source field, value is destination field or list of fields

classmethod `get_configuration()`

Find a SolrSchema subclass for use as schema configuration. Currently only supports one schema configuration.

classmethod `get_field_names()`

iterate over class attributes and return all that are instances of *SolrField*.

Return type

list

Returns

List of attributes that are *SolrField*.

classmethod `get_field_types()`

iterate over class attributes and return all that are instances of *SolrFieldType*.

Return type

list

Returns

List of attributes that are *SolrFieldType*.

class `parasolr.schema.SolrStringField(*args, **kwargs)`

Solr string field.

class `parasolr.schema.SolrTypedField(*args, **kwargs)`

Base class for typed solr field descriptor. For use with your own field types, extend and set `field_type`.

Parameters

- ***args** (*Any*) – Arguments as passed to *SolrField*.
- ****kwargs** (*Any*) – Keyword arguments as passed to *SolrField*.

1.3 Indexing

Model-based indexing with Solr.

Items to be indexed in Solr should extend *Indexable*. The default implementation should work for most Django models; at a minimum you should extend *Indexable.index_data()* to include the information to be indexed in Solr. You may also customize *Indexable.index_item_type()* and *Indexable.index_item_id()*.

To manually index content in Solr, see `index manage` command documentation.

class parasolr.indexing.Indexable

Mixin for objects that are indexed in Solr. Subclasses must implement *index_id* and *index* methods.

When implementing an Indexable subclass where *items_to_index* returns something like a generator, which does not expose either a *count* method or can be counted with *len*, for use with the Django index manage command you should implement *total_to_index* and return the number of items to be indexed.

ID_SEPARATOR = '.'

id separator for auto-generated index ids

classmethod all_indexables()

Find all *Indexable* subclasses for indexing. Ignore abstract and proxy *Indexable* subclasses such as *ModelIndexable*.

index()

Index the current object in Solr.

index_chunk_size = 150

number of items to index at once when indexing a large number of items

index_data()

Dictionary of data to index in Solr for this item. Default implementation adds *index_id()* and *index_item_type()*

index_id()

Solr identifier. By default, combines *index_item_type()* and *id* with `:attr:ID_SEPARATOR``.

classmethod index_item_type()

Label for this kind of indexable item. Must be unique across all Indexable items in an application. By default, uses Django model verbose name. Used in default index id and in index manage command.

classmethod index_items(*items*, *progressbar=None*)

Indexable class method to index multiple items at once. Takes a list, queryset, or generator of Indexable items or dictionaries. Items are indexed in chunks, based on *Indexable.index_chunk_size*.

Parameters

- **items** – list, queryset, or generator of indexable objects or dictionaries
- **progressbar** – optional `progressbar.ProgressBar` object to
- **chunks.** (*update when indexing items in*) –

Returns

Total number of items indexed

classmethod items_to_index()

Get all items to be indexed for a single class of Indexable content. Subclasses can override this method to return a custom iterable, e.g. a Django *QuerySet* that takes advantage of prefetching. By default, returns all Django objects for a model. Raises `NotImplementedError` if that fails.

classmethod prep_index_chunk(*chunk*)

Optional method for any additional processing on chunks of items being indexed. Intended to allow adding prefetching on a chunk when iterating on Django QuerySets; since indexing uses `Iterator`, prefetching configured in *items_to_index* is ignored.

remove_from_index()

Remove the current object from Solr by identifier using *index_id()*

solr = None

solr connection

classmethod total_to_index()

Get the total number of items to be indexed for a single class of Indexable content. Subclasses should override this method if necessary. By default, returns a Django queryset count for a model. Raises NotImplementedError if that fails.

parasolr.indexing.all_subclasses(cls)

recursive method to find all subclasses

1.4 QuerySet

Object-oriented approach to Solr searching and filtering modeled on `django.models.queryset.QuerySet`. Supports iteration, slicing, counting, and boolean check to see if a search has results.

Filter, search and sort methods return a new queryset, and can be chained. For example:

```
SolrQuerySet(solrclient).filter(item_type_s='person')
↪search(name='hem*') .order_by('sort_name')
```

If you are working with Django you should use `parasolr.django.SolrQuerySet`, which will automatically initialize a new `parasolr.django.SolrClient` if one is not passed in.

class parasolr.query.queryset.EmptySolrQuerySet(*args, **kwargs)

Marker class that can be used to check if a given queryset is empty via `isinstance()`:

```
assert isinstance(SolrQuerySet().none(), EmptySolrQuerySet) -> True
assert isinstance(queryset, EmptySolrQuerySet) # True if empty
```

class parasolr.query.queryset.SolrQuerySet(solr)

A Solr queryset object that allows for object oriented searching and filtering of Solr results. Allows search results to be pagination using slicing, count, and iteration.

ANY_VALUE = '[* TO *]'

any value constant

LOOKUP_SEP = '___'

lookup separator

_clone()

Return a copy of the current QuerySet for modification via filters.

Return type

SolrQuerySet

static _lookup_to_filter(key, value, tag='')

Convert keyword/value argument, with optional lookups separated by `___`, including: `in` and `exists`. Field names should *NOT* include double-underscores by convention. Accepts an optional tag argument to specify an exclude tag as needed. :rtype: `str`

Returns: A properly formatted Solr query string.

_set_faceting_opts(query_opts)

Configure faceting attributes directly on `query_opts`. Modifies dictionary directly.

Return type

`None`

`_set_group_opts(query_opts)`

Configure grouping attributes on `query_opts`. Modifies dictionary directly.

Return type

`None`

`_set_highlighting_opts(query_opts)`

Configure highlighting attributes on `query_opts`. Modifies dictionary directly.

Return type

`None`

`_set_stats_opts(query_opts)`

Configure stats attributes directly on `query_opts`. Modifies dictionary directly.

Return type

`None`

`all()`

Return a new queryset that is a copy of the current one.

Return type

`SolrQuerySet`

`also(*args, **kwargs)`

Use field limit option to return the specified fields, optionally provide aliases for them in the return. Works exactly the same way as `only()` except that it does not any previously specified field limits.

Return type

`SolrQuerySet`

`count()`

Total number of results for the current query

Return type

`int`

`default_search_operator = 'AND'`

by default, combine search queries with AND

`facet(*args, **kwargs)`

Request facets for specified fields. Returns a new `SolrQuerySet` with Solr faceting enabled and `facet.field` parameter set. Does not support ranged faceting.

Subsequent calls will reset the `facet.field` to the last set of args in the chain.

For example:

```
qs = queryset.facet('person_type', 'age')
qs = qs.facet('item_type_s')
```

would result in `item_type_s` being the only facet field.

Return type

`SolrQuerySet`

facet_field(*field*, *exclude=""*, ***kwargs*)

Request faceting for a single field. Returns a new SolrQuerySet with Solr faceting enabled and the field added to the list of facet fields. Any keyword arguments will be set as field-specific facet configurations.

ex will specify a related filter query tag to exclude when generating counts for the facet.

Return type

SolrQuerySet

facet_range(*field*, ***kwargs*)

Request range faceting for a single field. Returns a new SolrQuerySet with Solr range faceting enabled and the field added to the list of facet fields. Keyword arguments such as start, end, and gap will be set as field-specific facet configurations.

Return type

SolrQuerySet

filter(**args*, *tag=""*, ***kwargs*)

Return a new SolrQuerySet with Solr filter queries added. Multiple filters can be combined either in a single method call, or they can be chained for the same effect. For example:

```
queryset.filter(item_type_s='person').filter(birth_year=1900)
queryset.filter(item_type_s='person', birth_year=1900)
```

A tag may be specified for the filter to be used with facet.field exclusions:

```
queryset.filter(item_type_s='person', tag='person')
```

To provide a filter that should be used unmodified, provide the exact string of your filter query:

```
queryset.filter('birth_year:[1800 TO *]')
```

You can also search for pre-defined using lookups on a field, for example:

```
queryset.filter(item_type_s__in=['person', 'book'])
queryset.filter(item_type_s__exists=False)
```

Currently supported field lookups: *rtype: SolrQuerySet*

- **in** : takes a list of values; supports '' or None to match on field not set
- **exists**: boolean filter to look for any value / no value
- **range**: **range query. Takes a list or tuple of two values**
for the start and end of the range. Either value can be unset for an open-ended range (e.g. *year__range=(1800, None)*)

get_expanded()

Return a dictionary of expanded records included in the Solr response.

Return type

Dict[str, Dict]

get_facets()

Return a dictionary of facet information included in the Solr response. Includes facet fields, facet ranges, etc. Facet field results are returned as an ordered dict of value and count.

Return type

Dict[str, Dict]

get_highlighting()

Return the highlighting portion of the Solr response.

Return type

`Dict[str, Dict[str, List]]`

get_response(kwargs)**

Query Solr and get the results for the current query and filter options. Populates result cache and returns the documents portion of the reponse.

Return type

`List[dict]`

Returns

Solr response documents as a list of dictionaries.

get_result_document(doc)

Method to transform document results. Default behavior is to convert from attrdict to dict.

get_results(kwargs)**

Query Solr and get the results for the current query and filter options. Populates result cache and returns the documents portion of the reponse. (Note that this method is not currently compatible with grouping.)

Return type

`List[dict]`

Returns

Solr response documents as a list of dictionaries.

get_stats()

Return a dictionary of stats information in Solr format or None on error.

Return type

`Optional[Dict[str, ParasolrDict]]`

group(field, **kwargs)

“Configure grouping. Takes arbitrary Solr group parameters and adds the *group.* prefix to them. Example use, grouping on a *group_id* field, limiting to three results per group, and sorting group members by an *order* field:

```
queryset.group('group_id', limit=3, sort='order asc')
```

Return type

`SolrQuerySet`

highlight(field, **kwargs)

“Configure highlighting. Takes arbitrary Solr highlight parameters and adds the *hl.* prefix to them. Example use:

```
queryset.highlight('content', snippets=3, method='unified')
```

Return type

`SolrQuerySet`

none()

Return an empty result list.

Return type

SolrQuerySet

only(*args, replace=True, **kwargs)

Use field limit option to return only the specified fields. Optionally provide aliases for them in the return. Subsequent calls will *replace* any previous field limits. Example:

```
queryset.only('title', 'author', 'date')
queryset.only('title:title_t', 'date:pubyear_i')
```

Return type

SolrQuerySet

order_by(*args)

Apply sort options to the queryset by field name. If the field name starts with -, sort is descending; otherwise ascending.

Return type

SolrQuerySet

query(**kwargs)

Return a new SolrQuerySet with the results populated from Solr. Any options passed in via keyword arguments take precedence over query options on the queryset.

Return type

SolrQuerySet

query_opts()

Construct query options based on current queryset configuration. Includes filter queries, start and rows, sort, and search query.

Return type

Dict[str, str]

raw_query_parameters(**kwargs)

Add arbitrary raw parameters to be included in the query request, e.g. for variables referenced in join or field queries. Analogous to the input of the same name in the Solr web interface.

Return type

SolrQuerySet

search(*args, **kwargs)

Return a new SolrQuerySet with search queries added. All queries will combined with the default search operator when constructing the *q* parameter sent to Solr..

Return type

SolrQuerySet

set_limits(start, stop)

Set limits to get a subsection of the results, to support slicing.

stats(*args, **kwargs)

Request stats for specified fields. Returns a new SolrQuerySet with Solr faceting enabled and stats.field parameter set.

Subsequent calls will reset the stats.field to the last set of args in the chain.

For example:

```
qs = queryset.stats('person_type', 'age')
qs = qs.stats('account_start_i')
```

would result in `account_start_i` being the only facet field.

Any kwargs will be prepended with `stats..` You may also pass local parameters along with field names, i.e. `{!ex=filterA}account_start_i`.

Return type

SolrQuerySet

class `parasolr.query.aliased_queryset.AliasedSolrQuerySet(*args, **kwargs)`

Extension of *SolrQuerySet* with support for aliasing Solr fields to more readable versions for use in code. To use, extend this class and define a dictionary of *field_aliases* with the same syntax you would when calling *only()*. Those field aliases will be set as the default initial value for *field_list*, and aliases can be used in all extended methods.

_unalias_args(*args)

convert alias name to solr field for list of args

_unalias_kwargs(kwargs)**

convert alias name to solr field for keys in kwargs

_unalias_kwargs_with_lookups(kwargs)**

convert alias name to solr field for keys in kwargs with support for `__` lookups for filters

facet(*args, **kwargs)

Extend *parasolr.query.queryset.SolrQuerySet.facet()* to support using aliased field names in args.

Return type

AliasedSolrQuerySet

facet_field(field, exclude="", **kwargs)

Extend *parasolr.query.queryset.SolrQuerySet.facet_field`()* to support using aliased field names for field parameter.

Return type

AliasedSolrQuerySet

field_aliases = {}

map of application-specific, readable field names to actual solr fields (i.e. if using dynamic field types)

filter(*args, tag="", **kwargs)

Extend *parasolr.query.queryset.SolrQuerySet.filter()* to support using aliased field names for keyword argument keys.

Return type

AliasedSolrQuerySet

get_facets()

Extend *parasolr.query.queryset.SolrQuerySet.get_facets()* to use aliased field names for facet and range facet keys.

Return type

`Dict[str, int]`

get_highlighting()

Return the highlighting portion of the Solr response.

Return type

`Dict[str, Dict[str, List]]`

get_stats()

Extend `parasolr.query.queryset.SolrQuerySet.get_stats()` to return return aliased field names for `field_list` keys.

Return type

`Dict[str, Dict]`

group(field, **kwargs)

Extend `parasolr.query.queryset.SolrQuerySet.group()` to support using aliased field names in `kwargs`. (Note that sorting does not currently support aliased field names).

Return type

`AliasedSolrQuerySet`

highlight(field, **kwargs)

Extend `parasolr.query.queryset.SolrQuerySet.highlight()` to support using aliased field names in `kwargs`.

Return type

`AliasedSolrQuerySet`

only(*args, **kwargs)

Extend `parasolr.query.queryset.SolrQuerySet.only`()` to support using aliased field names for `args` (but not `kwargs`).

Return type

`AliasedSolrQuerySet`

order_by(*args)

Extend `parasolr.query.queryset.SolrQuerySet.order_by`()` to support using aliased field names in sort arguments.

Return type

`AliasedSolrQuerySet`

stats(*args, **kwargs)

Extend `parasolr.query.queryset.SolrQuerySet.stats()` to support using aliased field names in `args`.

Return type

`AliasedSolrQuerySet`

1.5 Django

1.5.1 Indexing

This module provides indexing support for Django models. Also see [Indexable](#).

To use, add `ModelIndexable` as a mixin to the model class you want to be indexed. At minimum, you'll want to extend the `index_data` method to include the data you want in the indexed:

```
def index_data(self):
    index_data = super().index_data()

    # if there are some records that should not be included
    # return id only. This will blank out any previously indexed
    # values, and item will not be findable by type.
    # if not ...
        # del index_data['item_type']
        # return index_data

    # add values to index data
    index_data.update({
        ...
    })
    return index_data
```

You can optionally extend `items_to_index()` and `index_item_type()`.

```
class parasolr.django.indexing.ModelIndexable(*args, **kwargs)
```

```
    static get_related_model(model, name)
```

Find a related model for use in signal-based indexing. Supports app.Model notation or attribute on the current model (supports queryset syntax for attributes on related models.)

```
    classmethod identify_index_dependencies()
```

Identify and set lists of index dependencies for the subclass of Indexable.

1.5.2 QuerySet

Provides SolrQuerySet subclasses that will automatically use SolrClient if no solr client is passed on.

```
class parasolr.django.queryset.AliasedSolrQuerySet(solr=None)
```

Combination of `SolrQuerySet` and `AliasedSolrQuerySet`

```
class parasolr.django.queryset.SolrQuerySet(solr=None)
```

SolrQuerySet subclass that will automatically use SolrClient if no solr client is passed on.

:param Optional `parasolr.solr.client.SolrClient`:

1.5.3 Signals

This module provides on-demand reindexing of Django models when they change, based on Django signals. To use this signal handler, import import it in the `ready` method of a django app. This will automatically bind connect any configured signal handlers:

```
from django.apps import AppConfig

class MyAppConfig(AppConfig):
    name = 'myapp'

    def ready(self):
```

(continues on next page)

(continued from previous page)

```
# import and connect signal handlers for Solr indexing
from parasolr.django.signals import IndexableSignalHandler
```

To configure index dependencies, add a property on any *ModelIndexable* subclass with the dependencies and signals that should trigger reindexing. Example:

```
class MyModel(ModelIndexable):

    index_depends_on = {
        'collections': {
            'post_save': signal_method,
            'pre_delete': signal_method
        }
    }
```

The keys of the dependency dict can be:

- an attribute on the indexable model (i.e., the name of a many-to-many relationship); this will bind an additional signal handler on the m2m relationship change.
- an attribute on a related model using django queryset notation (use this for a secondary many-to many relationship, e.g. *collections__authors*)
- a string with the model name in app.ModelName notation, to find and load a model directly

The dictionaries for each related model or attribute should contain:

- a key with the `django.db.models.signals` signal to bind
- a signal handler to bind

Currently attribute lookup only supports many-to-many and reverse many-to-many relationships.

Typically you will want to bind `post_save` and `pre_delete` for many-to-many relationships.

class `parasolr.django.signals.IndexableSignalHandler`

Signal handler for indexing Django model-based indexables. Automatically identifies and binds handlers based on configured index dependencies on indexable objects..

static `connect()`

bind indexing signal handlers to save and delete signals for *Indexable* subclassess and any indexing dependencies

static `disconnect()`

disconnect indexing signal handlers

static `handle_delete(sender, instance, **kwargs)`

remove from index on delete if an instance of *ModelIndexable*

static `handle_relation_change(sender, instance, action, **kwargs)`

index on add, remove, and clear for *ModelIndexable* instances

static `handle_save(sender, instance, **kwargs)`

reindex on save if an instance of *ModelIndexable*

1.5.4 Views

class parasolr.django.views.SolrLastModifiedMixin(**kwargs)

View mixin to add last modified headers based on Solr. By default, searches entire solr collection and returns the most recent last modified value (assumes **last_modified** field). To filter for items specific to your view, either set `solr_lastmodified_filters` or implement `get_solr_lastmodified_filters()`.

Constructor. Called in the URLconf; can contain helpful extra keyword arguments, and other things.

dispatch(request, *args, **kwargs)

Wrap the dispatch method to add a last modified header if one is available, then return a conditional response.

get_solr_lastmodified_filters()

Get filters for last modified Solr query. By default returns `solr_lastmodified_filters`.

last_modified()

Return last modified `datetime.datetime` from the specified Solr query

solr_lastmodified_filters = {}

solr query filter for getting last modified date

1.6 Manage Commands

1.6.1 Solr schema

solr_schema is a custom manage command to update the configured schema definition for the configured Solr instance. Reports on the number of fields that are added or updated, and any that are out of date and were removed.

Example usage:

```
python manage.py solr_schema
```

1.6.2 Index

GETTING STARTED

To use `parasolr` you need a [Solr installation](#) that you can connect to. Once you have Solr set up, use `solr start` to make sure it's running, and then create a new core: `solr create -c core_name`.

To interact with solr, use the `SolrClient` included in `parasolr`. It should be initialized with the URL for your Solr installation and the name of the core you want to query:

```
from parasolr.solr.client import SolrClient

solr_url = "http://localhost:8983/solr"
solr_core = "core_name"
solr = SolrClient(solr_url, solr_core)
```

Now you can index some data. The `index` method takes a list of dictionaries; note that any content you include must be JSON-serializable. For example, to index data from a CSV file:

```
solr.update.index([
    {"id": row["id"],
     "name": row["name"],
     "tags": row['tags'].split('|')
     # etc ...
} for row in csv])
```

To query the data you've indexed, initialize a `SolrQuerySet`, passing it the solr client you used before:

```
from parasolr.query import SolrQuerySet

queryset = SolrQuerySet(solr)
queryset = queryset.search('search string').order_by('name')
results = queryset.get_results(rows=20)
```

`results` contains a list of dictionaries that you're can manipulate or display as needed.

To remove records from your solr core, you can delete based on a query. For example, to delete all indexed items:

```
solr.update.delete_by_query('*:*')
```


CHANGELOG

3.1 0.9

- `SolrQuerySet` now supports Solr grouping via new `group` method and *GroupedResponse*
- New class method `prep_index_chunk` on `Indexable` class, to support prefetching related objects when iterating over Django querysets for indexing
- Include django view mixins in sphinx documentation
- Dropped support for python 3.6; added python 3.9
- Dropped support for Django 2.2; added Django 3.2
- No longer tested against Solr 6.6

3.2 0.8.2

- When subclassing `SolrQuerySet`, result documents can now be customized by extending `get_result_document`

3.3 0.8.1

- Exclude proxy models when collecting indexable subclasses

3.4 0.8

- Pytest fixture `mock_solr_queryset` now takes optional argument for extra methods to include in fluent interface
- `SolrQuerySet` now supports highlighting on multiple fields via `highlight` method, with per-field highlighting options.
- `AliasedSolrQuerySet` now correctly aliases fieldnames in highlighting results.
- Adopted black & isort python style and configured pre-commit hook

3.5 0.7

- Dropped support for Python 3.5
- Now tested against Python 3.6, 3.8, Django 2.2—3.1, Solr 6 and Solr 8
- Continuous integration migrated from Travis-CI to GitHub Actions
- bugfix: in some cases, index script was wrongly detecting `ModelIndexable` subclasses as abstract and excluding them; this has been corrected
- `ModelIndexable` now extends `django.db.models.Model`; existing code **MUST** be updated to avoid double-extending `Model`
- Default index data has been updated to use a dynamic field `item_type_s` instead of `item_type` so that basic setup does not require customizing the solr schema.
- `ModelIndexable.get_related_model` now supports `ForeignKey` relationships and `django-taggit TaggableManager` when identifying dependencies for binding signal handlers

3.6 0.6.1

- bugfix: fix regression in `SolrQuerySet` `get_stats` in 0.6

3.7 0.6

- Solr client now escalates 404 errors instead of logging with no exception
- Schema field declarations now support the `stored` option
- Schema field type declarations now pass through arbitrary options
- New method `total_to_index` on `parasolr.indexing.Indexable` to better support indexing content that is returned as a generator
- Access to expanded results now available on `QueryResponse` and `SolrQuerySet`
- `SolrQuerySet` no longer wraps return results from `get_stats` and `get_facets` with `QueryResponse`
- New last-modified view mixin for use with Django views `parasolr.django.views.SolrLastModifiedMixin`
- New pytest fixture `mock_solr_queryset` to generate a `Mock SolrQuerySet` that simulates the `SolrQuerySet` fluent interface

3.8 0.5.4

- Only enable pytest plugin when parasolr is in Django installed apps and a Solr connection is configured

0.5.3 —

- Support default option adding fields to solr schema
- Add utility method to convert Solr timestamp to python datetime

3.9 0.5.2

- bugfix: correct queryset highlighting so it actually works
- Revise pytest plugin code to work on non-django projects

3.10 0.5.1

- bugfix: SolrQuerySet improved handling for Solr errors

3.11 0.5

- Support for on-demand indexing for Django models based on signals; see `parasolr.django.signals`; adds a Django-specific indexable class `parasolr.django.indexing.ModelIndexable`
- pytest plugin to disconnect django signal handlers
- Django pytest fixture for an empty solr
- Adds an `EmptySolrQuerySet` class, as a simpler way to check for empty results

3.12 0.4

- `parasolr.query.SolrQuery` additional support for stats:
 - New method `stats` to enable stats for a set of field names.
 - New method `get_stats` to return the entire stats response.

3.13 0.3

- `parasolr.query.SolrQuerySet` additional support for faceting:
 - New method `facet_field` for more fine-grained facet feature control for a single facet field
 - New method `facet_range` for enabling range faceting
 - Supports tag and exclusion logic via `tag` option on `facet_field` method and `exclude` option on `filter`
 - `get_facets` now returns the entire facet response, including facet fields, range facets, etc.
- `SolrQuerySet.filter()` method now supports the following advanced lookups:
 - **in**: filter on a list of values
 - **exists**: filter on empty or not-empty
 - **range**: filter on a numeric range
- New method `SolrQuerySet.also()` that functions just like `only()` except it adds instead of replacing field limit options.
- New `parasolr.query.AliasedSolrQuerySet` supports aliasing Solr fields to local names for use across all queryset methods and return values

- `parasolr.indexing.Indexable` now provides `items_to_index()` method to support customizing retrieving items for indexing with `index manage` command.

3.14 0.2

- Subquent calls to `SolrQuerySet.only()` now *replaces* field limit options rather than adding to them.
- New `SolrQuerySet` method `raw_query_parameters`
- `SolrQuerySet` now has support for faceting via `facet` method to configure facets on the request and `get_facets` to retrieve them from the response.
- Update `ping` method of `parasolr.solr.admin.CoreAdmin` so that a 404 response is not logged as an error.
- Refactor `parasolr.solr` tests into submodules

3.15 0.1.1

- Fix travis-ci build for code coverage reporting.

3.16 0.1

Lightweight python library for Solr indexing, searching and schema management with optional Django integration.

- Minimal Python Solr API client
- Logic for updating and managing Solr schema
- Indexable mixin for Django models
- `QuerySet` for querying Solr in an object-oriented fashion similar to Django `QuerySet`
- Django Solr client with configuration from Django settings
- Django manage command to configure Solr schema
- Django manage command to index subclasses of `Indexable`
- `pytest` plugin for unit testing against a test Solr instance in Django
- Basic Sphinx documentation

parasolr is a lightweight python library for [Apache Solr](#) indexing, searching and schema management with optional [Django](#) integration. It includes a Solr client (`parasolr.solr.SolrClient`). When used with Django, it provides management commands for updating your Solr schema configuration and indexing content.

•

•

Currently tested against Python 3.8 and 3.9, Solr 8.6.2, and Django 3.0-3.2 and without Django.

INSTALLATION

Install released version from pypi:

```
pip install parasolr
```

To install an unreleased version from GitHub:

```
pip install git+https://github.com/Princeton-CDH/parasolr@develop#egg=parasolr
```

To use with Django:

- Add *parasolr* to **INSTALLED_APPS**
- Configure **SOLR_CONNECTIONS** in your django settings:

```
SOLR_CONNECTIONS = {
    'default': {
        'URL': 'http://localhost:8983/solr/',
        'COLLECTION': 'name',
        # Any configSet in SOLR_ROOT/server/solr/configsets.
        # The default configset name is "_default" as of Solr 7.
        # For Solr 6, "basic_configs" is the default.
        'CONFIGSET': '_default'
    }
}
```

- Define a *SolrSchema* with fields and field types for your project.
- Run `solr_schema manage` command to configure your schema; it will prompt to create the Solr core if it does not exist.

Note: The *SolrSchema* must be imported somewhere for it to be found automatically.

DEVELOPMENT INSTRUCTIONS

This git repository uses git flow branching conventions.

Initial setup and installation:

- *Recommended:* create and activate a Python 3.6 virtualenv:

```
python3 -m venv parasolr
source parasolr/bin/activate
```

- Install the package with its dependencies as well as development dependencies:

```
pip install -e .
pip install -e '[dev]'
```

5.1 Install pre-commit hooks

Install configured pre-commit hooks (currently `black` and `isort`):

```
pre-commit install
```

Styling was instituted in version 0.8; as a result, `git blame` may not reflect the true author of a given line. In order to see a more accurate `git blame` execute the following command:

```
git blame <FILE> --ignore-revs-file .git-blame-ignore-revs
```

Or configure your git to always ignore the black revision commit:

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```


UNIT TESTING

Unit tests are written with `pytest` but use some Django test classes for compatibility with Django test suites. Running the tests requires a minimal settings file for Django-required configurations.

- Copy sample test settings and add a secret key:

```
cp ci/testsettings.py testsettings.py
python -c "import uuid; print('\nSECRET_KEY = \'' + str(uuid.uuid4()) + '\')" >>>
↪testsettings.py
```

- By default, parasolr expects Solr 8. If running tests with an earlier version of Solr, either explicitly change **MAJOR_SOLR_VERSION** in your local **testsettings.py** or set the environment variable:

```
export SOLR_VERSION=x.x.x
```

- To run the test, either use the configured `setup.py` test command:

```
python setup.py test
```

- Or install test requirements in and use `pytest` directly:

```
pip install -e '.[test]'
pytest
```


LICENSE

parasolr is distributed under the Apache 2.0 License.

©2019 Trustees of Princeton University. Permission granted via Princeton Docket #20-3619 for distribution online under a standard Open Source license. Ownership rights transferred to Rebecca Koeser provided software is distributed online via open source.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- `parasolr.django`, [19](#)
- `parasolr.django.indexing`, [19](#)
- `parasolr.django.queryset`, [20](#)
- `parasolr.django.signals`, [20](#)
- `parasolr.django.views`, [22](#)
- `parasolr.indexing`, [11](#)
- `parasolr.management.commands.solr_schema`, [22](#)
- `parasolr.query.aliased_queryset`, [18](#)
- `parasolr.query.queryset`, [13](#)
- `parasolr.schema`, [8](#)
- `parasolr.solr.admin`, [7](#)
- `parasolr.solr.base`, [1](#)
- `parasolr.solr.client`, [2](#)
- `parasolr.solr.schema`, [4](#)
- `parasolr.solr.update`, [6](#)

Symbols

- `_clone()` (*parasolr.query.queryset.SolrQuerySet* method), 13
- `_delete()` (*parasolr.solr.update.Update* method), 6
- `_lookup_to_filter()` (*parasolr.query.queryset.SolrQuerySet* static method), 13
- `_post_field()` (*parasolr.solr.schema.Schema* method), 4
- `_process_facet_counts()` (*parasolr.solr.client.BaseResponse* method), 2
- `_set_faceting_opts()` (*parasolr.query.queryset.SolrQuerySet* method), 13
- `_set_group_opts()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `_set_highlighting_opts()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `_set_stats_opts()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `_unalias_args()` (*parasolr.query.aliased_queryset.AliasedSolrQuerySet* method), 18
- `_unalias_kwargs()` (*parasolr.query.aliased_queryset.AliasedSolrQuerySet* method), 18
- `_unalias_kwargs_with_lookups()` (*parasolr.query.aliased_queryset.AliasedSolrQuerySet* method), 18
- A**
- `add_copy_field()` (*parasolr.solr.schema.Schema* method), 4
- `add_field()` (*parasolr.solr.schema.Schema* method), 4
- `add_field_type()` (*parasolr.solr.schema.Schema* method), 4
- `AliasedSolrQuerySet` (class in *parasolr.django.queryset*), 20
- `AliasedSolrQuerySet` (class in *parasolr.query.aliased_queryset*), 18
- `all()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `all_indexables()` (*parasolr.indexing.Indexable* class method), 12
- `all_subclasses()` (in module *parasolr.indexing*), 13
- `also()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `ANY_VALUE` (*parasolr.query.queryset.SolrQuerySet* attribute), 13
- `as_dict()` (*parasolr.solr.client.ParasolrDict* method), 3
- B**
- `BaseResponse` (class in *parasolr.solr.client*), 2
- `build_url()` (*parasolr.solr.base.ClientBase* method), 1
- C**
- `ClientBase` (class in *parasolr.solr.base*), 1
- `collection` (*parasolr.solr.client.SolrClient* attribute), 3
- `commitWithin` (*parasolr.solr.client.SolrClient* attribute), 3
- `configure_copy_fields()` (*parasolr.schema.SolrSchema* class method), 10
- `configure_fields()` (*parasolr.schema.SolrSchema* class method), 10
- `configure_fieldtypes()` (*parasolr.schema.SolrSchema* class method), 10
- `connect()` (*parasolr.django.signals.IndexableSignalHandler* static method), 21
- `copy_fields` (*parasolr.schema.SolrSchema* attribute), 11
- `core_admin_handler` (*parasolr.solr.client.SolrClient* attribute), 3
- `CoreAdmin` (class in *parasolr.solr.admin*), 7
- `CoreExists`, 2
- `count()` (*parasolr.query.queryset.SolrQuerySet* method), 14
- `create()` (*parasolr.solr.admin.CoreAdmin* method), 7

D

`default_search_operator` (parasolr.query.queryset.SolrQuerySet attribute), 14

`delete_by_id()` (parasolr.solr.update.Update method), 6

`delete_by_query()` (parasolr.solr.update.Update method), 7

`delete_copy_field()` (parasolr.solr.schema.Schema method), 4

`delete_field()` (parasolr.solr.schema.Schema method), 5

`delete_field_type()` (parasolr.solr.schema.Schema method), 5

`disconnect()` (parasolr.django.signals.IndexableSignalHandler static method), 21

`dispatch()` (parasolr.django.views.SolrLastModifiedMixin method), 22

E

`EmptySolrQuerySet` (class in parasolr.query.queryset), 13

`expanded` (parasolr.solr.client.BaseResponse property), 2

F

`facet()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 18

`facet()` (parasolr.query.queryset.SolrQuerySet method), 14

`facet_field()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 18

`facet_field()` (parasolr.query.queryset.SolrQuerySet method), 14

`facet_range()` (parasolr.query.queryset.SolrQuerySet method), 15

`field_aliases` (parasolr.query.aliased_queryset.AliasedSolrQuerySet attribute), 18

`filter()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 18

`filter()` (parasolr.query.queryset.SolrQuerySet method), 15

`filters` (parasolr.schema.SolrAnalyzer attribute), 9

G

`get_configuration()` (parasolr.schema.SolrSchema class method), 11

`get_expanded()` (parasolr.query.queryset.SolrQuerySet method), 15

`get_facets()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 18

`get_facets()` (parasolr.query.queryset.SolrQuerySet method), 15

`get_field_names()` (parasolr.schema.SolrSchema class method), 11

`get_field_types()` (parasolr.schema.SolrSchema class method), 11

`get_highlighting()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 18

`get_highlighting()` (parasolr.query.queryset.SolrQuerySet method), 15

`get_related_model()` (parasolr.django.indexing.ModelIndexable static method), 20

`get_response()` (parasolr.query.queryset.SolrQuerySet method), 16

`get_result_document()` (parasolr.query.queryset.SolrQuerySet method), 16

`get_results()` (parasolr.query.queryset.SolrQuerySet method), 16

`get_schema()` (parasolr.solr.schema.Schema method), 5

`get_solr_lastmodified_filters()` (parasolr.django.views.SolrLastModifiedMixin method), 22

`get_stats()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 19

`get_stats()` (parasolr.query.queryset.SolrQuerySet method), 16

`group()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 19

`group()` (parasolr.query.queryset.SolrQuerySet method), 16

`group_field` (parasolr.solr.client.GroupedResponse property), 2

`GroupedResponse` (class in parasolr.solr.client), 2

`groups` (parasolr.solr.client.GroupedResponse property), 2

H

`handle_delete()` (parasolr.django.signals.IndexableSignalHandler static method), 21

`handle_relation_change()` (parasolr.django.signals.IndexableSignalHandler static method), 21

`handle_save()` (parasolr.django.signals.IndexableSignalHandler static method), 21

`highlight()` (parasolr.query.aliased_queryset.AliasedSolrQuerySet method), 19

`highlight()` (*parasolr.query.queryset.SolrQuerySet* method), 16
`highlighting` (*parasolr.solr.client.BaseResponse* property), 2

I

`ID_SEPARATOR` (*parasolr.indexing.Indexable* attribute), 12
`identify_index_dependencies()` (*parasolr.django.indexing.ModelIndexable* class method), 20
`ImproperConfiguration`, 2
`index()` (*parasolr.indexing.Indexable* method), 12
`index()` (*parasolr.solr.update.Update* method), 7
`index_chunk_size` (*parasolr.indexing.Indexable* attribute), 12
`index_data()` (*parasolr.indexing.Indexable* method), 12
`index_id()` (*parasolr.indexing.Indexable* method), 12
`index_item_type()` (*parasolr.indexing.Indexable* class method), 12
`index_items()` (*parasolr.indexing.Indexable* class method), 12
`Indexable` (class in *parasolr.indexing*), 11
`IndexableSignalHandler` (class in *parasolr.django.signals*), 21
`items_to_index()` (*parasolr.indexing.Indexable* class method), 12

L

`last_modified()` (*parasolr.django.views.SolrLastModifiedMixin* method), 22
`list_copy_fields()` (*parasolr.solr.schema.Schema* method), 5
`list_field_types()` (*parasolr.solr.schema.Schema* method), 5
`list_fields()` (*parasolr.solr.schema.Schema* method), 6
`LOOKUP_SEP` (*parasolr.query.queryset.SolrQuerySet* attribute), 13

M

`make_request()` (*parasolr.solr.base.ClientBase* method), 1
`ModelIndexable` (class in *parasolr.django.indexing*), 20
module
 parasolr.django, 19
 parasolr.django.indexing, 19
 parasolr.django.queryset, 20
 parasolr.django.signals, 20
 parasolr.django.views, 22
 parasolr.indexing, 11
 parasolr.management.commands.solr_schema, 22

parasolr.query.aliaseset, 18
parasolr.query.queryset, 13
parasolr.schema, 8
parasolr.solr.admin, 7
parasolr.solr.base, 1
parasolr.solr.client, 2
parasolr.solr.schema, 4
parasolr.solr.update, 6

N

`none()` (*parasolr.query.queryset.SolrQuerySet* method), 16

O

`only()` (*parasolr.query.aliaseset.AliasedSolrQuerySet* method), 19
`only()` (*parasolr.query.queryset.SolrQuerySet* method), 17
`order_by()` (*parasolr.query.aliaseset.AliasedSolrQuerySet* method), 19
`order_by()` (*parasolr.query.queryset.SolrQuerySet* method), 17

P

`params` (*parasolr.solr.client.BaseResponse* property), 2
parasolr.django
 module, 19
 parasolr.django.indexing
 module, 19
 parasolr.django.queryset
 module, 20
 parasolr.django.signals
 module, 20
 parasolr.django.views
 module, 22
parasolr.indexing
 module, 11
parasolr.management.commands.solr_schema
 module, 22
parasolr.query.aliaseset
 module, 18
parasolr.query.queryset
 module, 13
parasolr.schema
 module, 8
parasolr.solr.admin
 module, 7
parasolr.solr.base
 module, 1
parasolr.solr.client
 module, 2
parasolr.solr.schema
 module, 4
parasolr.solr.update

module, 6
 ParasoDict (class in parasolr.solr.client), 3
 ping() (parasolr.solr.admin.CoreAdmin method), 7
 prep_index_chunk() (parasolr.indexing.Indexable
 class method), 12

Q

query() (parasolr.query.queryset.SolrQuerySet
 method), 17
 query() (parasolr.solr.client.SolrClient method), 3
 query_opts() (parasolr.query.queryset.SolrQuerySet
 method), 17
 QueryResponse (class in parasolr.solr.client), 3

R

raw_query_parameters() (para-
 solr.query.queryset.SolrQuerySet method),
 17
 reload() (parasolr.solr.admin.CoreAdmin method), 8
 remove_from_index() (parasolr.indexing.Indexable
 method), 12
 replace_field() (parasolr.solr.schema.Schema
 method), 6
 replace_field_type() (parasolr.solr.schema.Schema
 method), 6

S

Schema (class in parasolr.solr.schema), 4
 schema_handler (parasolr.solr.client.SolrClient at-
 tribute), 3
 search() (parasolr.query.queryset.SolrQuerySet
 method), 17
 select_handler (parasolr.solr.client.SolrClient at-
 tribute), 3
 set_limits() (parasolr.query.queryset.SolrQuerySet
 method), 17
 solr (parasolr.indexing.Indexable attribute), 12
 solr_lastmodified_filters (para-
 solr.django.views.SolrLastModifiedMixin
 attribute), 22
 SolrAnalyzer (class in parasolr.schema), 9
 SolrClient (class in parasolr.solr.client), 3
 SolrClientException, 2
 SolrConnectionNotFound, 2
 SolrField (class in parasolr.schema), 10
 SolrFieldType (class in parasolr.schema), 10
 SolrLastModifiedMixin (class in para-
 solr.django.views), 22
 SolrQuerySet (class in parasolr.django.queryset), 20
 SolrQuerySet (class in parasolr.query.queryset), 13
 SolrSchema (class in parasolr.schema), 10
 SolrStringField (class in parasolr.schema), 11
 SolrTypedField (class in parasolr.schema), 11
 stats (parasolr.solr.client.BaseResponse property), 2

stats() (parasolr.query.aliased_queryset.AliasedSolrQuerySet
 method), 19
 stats() (parasolr.query.queryset.SolrQuerySet
 method), 17
 status() (parasolr.solr.admin.CoreAdmin method), 8

T

tokenizer (parasolr.schema.SolrAnalyzer attribute), 9
 total_to_index() (parasolr.indexing.Indexable class
 method), 13

U

unload() (parasolr.solr.admin.CoreAdmin method), 8
 Update (class in parasolr.solr.update), 6