Kaiso Documentation

Release 0.1-dev

onefinestay

Sep 27, 2017

Contents

A graph based queryable object persistance framework built on top of Neo4j.

CHAPTER 1

Neo4j visualization style

The bookmarklet below adds Kaiso styles to the Neo4j data browser's visualizer.

Apply kaiso-neo4j style

Add it to your bookmarks and load it when you're inside Neo's webadmin. A `kaiso` profile will become available under the Style button.

CHAPTER 2

Contents

API Reference

kaiso Package

A graph based query and persistance framework for objects and their relationships.

TODO: describe the architecture

connection Module

descriptors Module

exceptions Module

Provides all the exceptions that may be raised.

exception kaiso.exceptions.**CannotUpdateType**
Bases: exceptions.Exception

Raised when trying to update a type defined in code

exception kaiso.exceptions.**ConnectionError**
Bases: exceptions.Exception

Raised when an error occurs connecting to the Neo4j Database

exception kaiso.exceptions.**DeserialisationError**
Bases: exceptions.Exception

Raised when trying to deserialise a dict with no __type__ key

exception kaiso.exceptions.**MultipleObjectsFound**
Bases: exceptions.Exception

Raised when a caller of a RelationshipManager expected a single object, but multiple were returned.

exception kaiso.exceptions.**NoResultFound**
Bases: exceptions.Exception

Raised when a call expected at least one object, but none was found.

exception kaiso.exceptions.**NoUniqueAttributeError**
Bases: exceptions.Exception

Raised when trying to uniquely identify an object which has no unique attributes

exception kaiso.exceptions.**TypeAlreadyCollected**
Bases: exceptions.Exception

exception kaiso.exceptions.**TypeAlreadyRegistered**
Bases: exceptions.Exception

exception kaiso.exceptions.**TypeNotPersistedError**(*type_id*)
Bases: exceptions.Exception

Raised when trying to save an instance of a type that is not yet persisted

exception kaiso.exceptions.**UnknownType**
Bases: exceptions.Exception

Raised when trying to deserialise a class that hasn't been registered

exception kaiso.exceptions.**UnsupportedTypeError**
Bases: exceptions.Exception

Raised when trying to interact with a non-Persistable type

iter_helpers Module

kaiso.iter_helpers.**unique**(*fn*)
Wraps a function to return only unique items. The wrapped function must return an iterable object. When the wrapped function is called, each item from the iterable will be yielded only once and duplicates will be ignored.

Args: fn: The function to be wrapped.

Returns: A wrapper function for fn.

persistence Module

class kaiso.persistence.**Manager**(*connection_uri*, *skip_setup=False*)
Bases: object

Manage the interface to graph-based queryable object store.

The any object can be saved as long as its type is registered. This includes instances of Entity, PersistableType and subclasses of either.

InstanceOf and IsA relationships are automatically generated when persisting an object.

change_instance_type(*obj*, *type_id*, *updated_values=None*)

create_type(*name*, *bases*, *attrs*)
Creates a new class given the name, bases and attrs given.

delete(*obj*)
Deletes an object from the store.

Args: obj: The object to delete.

Returns: A tuple: with (number of nodes removed, number of rels removed)

deserialize(*object_dict*)
Deserialize `object_dict` to an object.

Args: object_dict: A serialized object dictionary

Returns: An object deserialized using the type registry

destroy()
Removes all nodes, relationships and indexes in the store. This object will no longer be usable after calling this method. Construct a new Manager to re-initialise the database for kaiso.

WARNING: This will destroy everything in your Neo4j database.

get(*cls*, ***attr_filter*)

get_by_unique_attr(*cls*, *attr_name*, *values*)
Bulk load entities from a list of values for a unique attribute

Returns: A generator (obj1, obj2, ...) corresponding to the *values* list

If any values are missing in the index, the corresponding obj is None

get_related_objects(*rel_cls*, *ref_cls*, *obj*)

get_type_hierarchy(*start_type_id=None*)
Returns the entire type hierarchy defined in the database if start_type_id is None, else returns from that type.

Returns: A generator yielding tuples of the form `(type_id, bases, attrs)` where

- `type_id` identifies the type
- `bases` lists the type_ids of the type's bases
- `attrs` lists the attributes defined on the type

invalidate_type_system()

query(*query*, ***params*)
Queries the store given a parameterized cypher query.

Args: query: A parameterized cypher query. params: query: A parameterized cypher query.

Returns: A generator with tuples containing stored objects or values.

WARNING: If you use this method to modify the type hierarchy (i.e. types, their declared attributes or their relationships), ensure to call `manager.invalidate_type_hierarchy()` afterwards. Otherwise managers will continue to use cached versions. Instances can be modified without changing the type hierarchy.

query_single(*query*, ***params*)
Convenience method for queries that return a single item

reload_types()
Reload the type registry for this instance from the graph database.

save(*persistable*)
Stores the given `persistable` in the graph database. If a matching object (by unique keys) already exists, it will update it with the modified attributes.

save_collected_classes(*collection*)

serialize(*obj*, *for_db=False*)
Serialize `obj` to a dictionary.

Args: obj: An object to serialize for_db: (Optional) bool to indicate whether we are serializing

data for neo4j or for general transport. This flag propagates down all the way into `Attribute.to_primitive` and may be used by custom attributes to determine behaviour for different serialisation targets. E.g. if using a transport that supports a Decimal type, *to_primitive* can return Decimal objects if for_db is False, and strings otherwise (for persistance in the neo4j db).

Returns: A dictionary describing the object

update_type(*tpe*, *bases*)
Change the bases of the given `tpe`

class kaiso.persistence.**TypeSystem**(**args*, ***kwargs*)
Bases: *kaiso.types.AttributedBase*

`TypeSystem` is a node that represents the root of the type hierarchy.

Inside the database, the current version of the hierarchy is tracked using a `version` attribute on the TypeSystem node.

id = <kaiso.attributes.String object>

kaiso.persistence.**get_connection**(*uri*)

references Module

kaiso.references.**get_store_for_object**(*obj*)

kaiso.references.**set_store_for_object**(*obj*, *store*)

relationships Module

Provides Relationship classes to be used by client code.

All relationship classes must inherit from Relationship to be usable within the framework.

The IsA and InstanceOf relationships are used internally for describing inheritance relationships and are autogenerated when an object is added to the data store.

They may be used for querying and map to pyhton's issubclass() and isinstanceof() functionality.

class kaiso.relationships.**DeclaredOn**(*start=None*, *end=None*, ***kwargs*)
Bases: *kaiso.types.Relationship*

Describes the relationship between a type and the properties that instances of that type can have.

class kaiso.relationships.**Defines**(*start=None*, *end=None*, ***kwargs*)
Bases: *kaiso.types.Relationship*

Relationship used by the `TypeSystem` to connect to the types it defines.

class kaiso.relationships.**InstanceOf**(*start=None*, *end=None*, ***kwargs*)
Bases: *kaiso.types.Relationship*

Describes a instance to type relationship between an object and a class.

It maps to python's isinstance(obj, the_class) as InstanceOf(obj, the_class).

class kaiso.relationships.**IsA**(*start=None*, *end=None*, ***kwargs*)
Bases: *kaiso.types.Relationship*

Describes a sub-class relationship between two classes.

It maps to python's issubclass(sub_class, base_class) as IsA(sub_class, base_class).

base_index = <kaiso.attributes.Integer object>

types Module

class kaiso.types.**Attribute**(*unique=False*, *required=False*)
Bases: *kaiso.types.AttributeBase*

required = None

unique = None

class kaiso.types.**AttributeBase**
Bases: object

name = None

classmethod to_primitive(*value*, *for_db*)
Serialize value to a primitive type suitable for inserting into the database or passing to e.g. json.dumps

classmethod to_python(*value*)

class kaiso.types.**AttributedBase**(**args*, ***kwargs*)
Bases: *kaiso.types.Persistable*

The base class for objects that can have Attributes.

Sets default values during instance creation and applies kwargs passed to __init__.

class kaiso.types.**DefaultableAttribute**(*default=None*, ***kwargs*)
Bases: *kaiso.types.Attribute*

default = None

class kaiso.types.**Descriptor**(*cls*)
Bases: object

Provides information about the types of persistable objects.

Its main purpose is to provide type names and attribute information of persistable types(classes).

attributes

class_attributes

declared_attributes

declared_class_attributes

relationships

class kaiso.types.**Entity**(**args*, ***kwargs*)
Bases: *kaiso.types.AttributedBase*

class kaiso.types.**Persistable**
Bases: object

The base of all persistable objects.

Any object stored in the db must inherit from this class. Any object having Persistable as it's base are considered persistable.

class kaiso.types.**PersistableType**
Bases: type, *kaiso.types.Persistable*

Metaclass for static persistable types.

Collects classes as they are declared so that they can be registered with the TypeRegistry later.

Collection can be controlled using the `collector` context manager.

class kaiso.types.**Relationship**(*start=None*, *end=None*, ***kwargs*)
Bases: *kaiso.types.AttributedBase*

class kaiso.types.**StaticClassCollector**
Bases: object

add_class(*cls*)

dump_state()

get_classes()

get_descriptors()

get_relationships()

load_state(*state*)

reset_state()

class kaiso.types.**TypeRegistry**
Bases: object

Keeps track of statically and dynamically declared types.

clone()
Return a copy of this TypeRegistry that maintains an independent dynamic type registry

create_type(*cls_id*, *bases*, *attrs*)
Create and register a dynamic type

dict_to_object(*properties*)
Converts a dict into a persistable object.

The properties dict needs at least a __type__ key containing the name of any registered class. The type key defines the type of the object to return.

If the registered class for the __type__ is a meta-class, i.e. a subclass of <type>, a name key is assumed to be present and the registered class idendified by it's value is returned.

If the registered class for the __type__ is standard class, i.e. an instance of <type>, and object of that class will be created with attributes as defined by the remaining key-value pairs.

Args: properties: A dict like object.

Returns: A persistable object.

get_class_by_id(*cls_id*)
Return the class for a given `cls_id`, preferring statically registered classes.

Returns the statically registered class with the given `cls_id`, if one exists, and otherwise returns any dynamically registered class with that id.

Arguments: cls_id: id of the class to return registry: type of registered class to prefer

Returns: The class that was registered with `cls_id`

get_constraints_for_type(*cls*)

get_descriptor(*cls*)

get_descriptor_by_id(*cls_id*)
Return the Descriptor for a given `cls_id`.

Returns the descriptor for the class registered with the given `cls_id`. If dynamic types have not been loaded yet, return the descriptor of the statically registered class.

Arguments: cls_id: id of the class

Returns: The Descriptor for that cls_id

Raises: UnknownType if no type has been registered with the given id.

get_labels_for_type(*cls*)
We set labels for any unique attributes

get_relationship_type_id(*neo4j_rel_name*)

get_unique_attrs(*cls*)
Generates tuples (declaring_class, attribute_name) for unique attributes

has_code_defined_attribute(*cls*, *attr_name*)
Determine whether an attribute called `attr_name` was defined in code on `cls` or one of its parent types.

is_static_type(*cls*)

object_to_dict(*obj*, *for_db=False*)
Converts a persistable object to a dict.

The generated dict will contain a __type__ key, for which the value will be the type_id as given by the descriptor for type(obj).

If the object is a class, dict will contain an id-key with the value being the type_id given by the descriptor for the object.

For any other object all the attributes as given by the object's type descriptor will be added to the dict and encoded as required.

Args: obj: A persistable object.

Returns: Dictionary with attributes encoded in basic types and type information for deserialization. e.g.
{

'__type__': 'Entity', 'attr1' : 1234

}

refresh_type(*cls*)

register(*cls*)
Register a dynamic type

kaiso.types.**cache_result**(*func*)
Cache the result of a function in self._cache

kaiso.types.**collector**(**args*, ***kwds*)
Allows code-defined types to be collected into a custom dict.

Example:

with collector() as type_map:

class Foobar(Entity): pass

type_map == {'Foobar': Foobar}

kaiso.types.**get_declaring_class**(*cls*, *attr_name*, *prefer_subclass=True*)
Returns the class in the type heirarchy of `cls` that defined an attribute with name `attr_name`.

If `prefer_subclass` is false, return the last class in the MRO that defined an attribute with the given name.

kaiso.types.**get_neo4j_relationship_name**(*cls*)

kaiso.types.**get_type_id**(*cls*)
Returns the type_id for a class.

Subpackages

attributes Package

attributes Package

class kaiso.attributes.**Bool**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of `bool`

class kaiso.attributes.**Choice**(**choices*, ***kwargs*)
Bases: *kaiso.types.DefaultableAttribute*

choices = <kaiso.attributes.Tuple object>

class kaiso.attributes.**DateTime**(*default=None*, ***kwargs*)
Bases: *kaiso.types.DefaultableAttribute*

classmethod to_primitive(*value*, *for_db*)

classmethod to_python(*value*)

class kaiso.attributes.**Decimal**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of `str`

classmethod to_python(*value*)

class kaiso.attributes.**Float**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of `float`

class kaiso.attributes.**Incoming**(*relationship_class*)
Bases: *kaiso.attributes.bases.RelationshipReference*

class kaiso.attributes.**Integer**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of `int`

class kaiso.attributes.**Outgoing**(*relationship_class*)
Bases: *kaiso.attributes.bases.RelationshipReference*

class kaiso.attributes.**PrimitiveTypeMixin**
Bases: object

Add a basic *to_primitive* method, coercing value to *cls.primitive_type* unless it is *None*.

classmethod to_primitive(*value*, *for_db*)

class kaiso.attributes.**String**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of unicode

class kaiso.attributes.**Tuple**(*default=None*, ***kwargs*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.DefaultableAttribute*

primitive_type
alias of list

classmethod to_python(*value*)

class kaiso.attributes.**Uuid**(*unique=False*, *required=False*)
Bases: *kaiso.attributes.PrimitiveTypeMixin*, *kaiso.types.Attribute*

default

primitive_type
alias of str

classmethod to_python(*value*)

bases Module

class kaiso.attributes.bases.**RelationshipManager**(*obj*, *relationship_reference*)
Bases: object

first()

one()

relationships

class kaiso.attributes.bases.**RelationshipReference**(*relationship_class*)
Bases: object

get_manager(*obj*)

kaiso.attributes.bases.**get_attibute_for_type**(*cls*)

kaiso.attributes.bases.**wraps_type**(*cls*)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

U

W