chide Documentation

Release 2.1.2

Chris Withers

Jan 31, 2019

Contents

Quickly create sample objects from data.

CHAPTER 1

Usage

Here's how to simply and quickly create sample objects using *chide*. For the examples, these two classes will be used:

```
class ClassOne(object):

  def __init__(self, x, y):
      self.x, self.y = x, y

class ClassTwo(object):

  def __init__(self, a, b):
      self.a, self.b = a, b
```

1.1 Creating sample objects

We can set up a registry of sample values as follows:

```
from chide import Collection

samples = Collection({
    ClassOne: {'x': 1, 'y': 2},
    ClassTwo: {'a': 1, 'b': ClassOne},
})
```

Now we can quickly make sample objects:

```
>>> samples.make(ClassOne)
<ClassOne ...>
>>> _.x, _.y
(1, 2)
```

We can provide our own overrides if we want:

```
>>> samples.make(ClassOne, y=3)
<ClassOne ...>
>>> _.x, _.y
(1, 3)
```

We can also create nested trees of objects:

```
>>> samples.make(ClassTwo)
<ClassTwo ...>
>>> _.b
<ClassOne ...>
```

These can also be overridden with another sample object:

```
>>> samples.make(ClassTwo, b=samples.make(ClassOne, x=-1))
<ClassTwo ...>
>>> _.b.x
-1
```

They can also be directly overridden with an appropriate instance:

```
>>> samples.make(ClassTwo, b=ClassOne(11, 3))
<ClassTwo ...>
>>> _.b.x, _.b.y
(11, 3)
```

1.2 Sets of unique sample objects

In some circumstances, you may need a set of related objects where the objects have a notion of an identifier, and only one object of each type with a given identifier can exist. Here's an oversimplified example:

```
class Person(object):
    def __init__(self, name, address):
        self.name = name
        self.address = address

class Address(object):
    seen = set()
    def __init__(self, value):
        if value in self.seen:
            raise Exception(value)
        self.seen.add(value)
        self.value = value
```

So, given that there can be multiple people but each address can only be instantiated once we want all people used in tests, by default, to be at the same address. We can do this using a *Set* and an *identify* function as follows:

```
from chide import Collection, Set

data = Collection({
    Person: {'name': 'Fred', 'address': Address},
    Address: {'value': 'some place in the clouds'},
})
```

(continues on next page)

(continued from previous page)

```
def identify(type_, attrs):
    if type_ is Address:
        return attrs['value']

samples = Set(data, identify)
```

Now, we can create multiple sample people without having to specify their address:

```
>>> person1 = samples.get(Person, name='Chris')
>>> person2 = samples.get(Person, name='Kirsty')
```

They both just end up at the same address:

```
>>> person1.address.value
'some place in the clouds'
>>> person2.address.value
'some place in the clouds'
>>> person1.address is person2.address
True
```

We can still create people with different addresses:

```
>>> person3 = samples.get(Person, name='Fred', address=Address('elsewhere'))
>>> person3.address.value
'elsewhere'
```

The *Set* can also be used to make sure that only one of each address is used:

```
>>> person4 = samples.get(Person, name='Bob', address=samples.get(Address, value='here
↪'))
>>> person5 = samples.get(Person, name='Joe', address=samples.get(Address, value='here
↪'))
```

This way, we don't have to manually ensure that only one address for *here* exists.

You'll notice that we can still get multiple people with the same name from the *Set*:

```
>>> person6 = samples.get(Person, name='Chris')
>>> person1 is person6
False
```

This because the `identify()` function above returns `None` for all types other than `Address`. Returning `None` from `identify()` is the way to indicate that a new object should be returned, regardless of the attributes it has.

1.3 Creating attributes for objects

Given this collection:

```
from chide import Collection

samples = Collection({
    ClassOne: {'x': 1, 'y': 2},
    ClassTwo: {'a': 1, 'b': ClassOne},
})
```

We can also create attributes to make a sample object:

```
>>> attrs = samples.attributes(ClassOne)
>>> attrs['x']
1
>>> attrs['y']
2
```

```
>>> attrs = samples.attributes(ClassTwo)
>>> attrs['a']
1
>>> attrs['b']
<ClassOne object at ...>
```

CHAPTER 2

Use with SQLAlchemy

chide has a special *Set* subclass that helps to make sure only one sample object is created with a particular primary key in any one table.

For example, given these two models:

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    child_id = Column(Integer, ForeignKey('child.id'))
    child = relationship('Child')

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    value = Column(Integer)
```

We can set up a collection of sample values as follows:

```
from chide import Collection

samples = Collection({
    Parent: {'id': 1, 'child': Child},
    Child: {'id': 3, 'value': 42}
})
```

Now we can quickly make sample objects and add them to a session:

```
>>> session = Session()
>>> session.add(samples.make(Parent))
>>> session.commit()
```

This gives us a parent and a child:

```
>>> session.query(Parent).one()
<Parent ...>
>>> _.child
<Child ...>
```

If we create multiple parent objects and don't want to have to worry about clashing children being created by mistake, we can use a `chide.sqlalchemy.Set` to make sure that we only have one sample object with a given primary key at any time:

```
>>> from chide.sqlalchemy import Set
>>> current_samples = Set(samples)
>>> session = Session()
>>> session.add(current_samples.get(Parent, id=1))
>>> session.add(current_samples.get(Parent, id=2))
>>> session.commit()
```

This gives us two parents that both point to the same child:

```
>>> parent1 = session.query(Parent).filter_by(id=1).one()
>>> parent2 = session.query(Parent).filter_by(id=2).one()
>>> parent1.child is parent2.child
True
```

Of course, if we want different children, that's easy too:

```
>>> session = Session()
>>> session.add(current_samples.get(Parent, id=3, child=Child(value=6)))
>>> session.add(current_samples.get(Parent, id=4, child=Child(value=7)))
>>> session.commit()
```

The children's primary keys will be created by the database, but the values are as we need them:

```
>>> parent3 = session.query(Parent).filter_by(id=3).one()
>>> parent3.child.value
6
>>> parent4 = session.query(Parent).filter_by(id=4).one()
>>> parent4.child.value
7
```

CHAPTER 3

API Reference

class chide.**Collection**(*mapping*)

A collection of attributes to use to make sample objects.

Parameters **mapping** – A dictionary mapping object types to a dictionary of attributes to make a sample object of that type.

attributes(*type_*, ***attrs*)

Make a sample object of the specified `type_` using the default attributes for that type in this *Collection*.

The `attrs` mapping will be overlayed onto the sample attributes before being used with `type_` to instantiate and return a new sample object.

make(*type_*, ***attrs*)

Make the attributes for a sample object of the specified `type_` using the default attributes for that type in this *Collection*.

The `attrs` mapping will be overlayed onto the sample attributes and returned as a `dict`.

class chide.**Set**(*collection*, *identify=None*)

A collection of sample objects where only one object with a given identity may exist at one time.

Parameters

- **collection** – The *Collection* instance used to create sample objects when necessary.
- **identify** – A callable that takes *type_* and *attrs* parameters.

  *type_*, usually a class, is the type of the sample object being requested.

  attrs is a `dict` of the attributes being requested for the sample object to have.

  The callable should return a hashable value that indicates the identity to use for the requested sample object. For each unique hashable value, only one sample object will be instantiated and returned each time a sample is requested where this callable returns the given identity.

  `None` may be returned to indicate that a new object should always be returned for the provided parameters.

get(*type_*, ***attrs*)
Return an appropriate sample object of the specified `type_`.

The `attrs` mapping will be overlaid onto the sample attributes found in this set's *`Collection`* before checking if an appropriate sample object already exists in the set.

If one exists, it is returned. If not, one is created using this set's *`Collection`*, added to the set and then returned.

identify = None
You may also want to subclass *`Set`* and implement an *`identify()`* method, see `chide.sqlalchemy.Set` for an example.

CHAPTER 4

Development

This package is developed using continuous integration which can be found here:

https://travis-ci.org/cjw296/chide

The latest development version of the documentation can be found here:

http://chide.readthedocs.org/en/latest/

If you wish to contribute to this project, then you should fork the repository found here:

https://github.com/cjw296/chide/

Once that has been done and you have a checkout, you can follow these instructions to perform various development tasks:

4.1 Setting up a virtualenv

The recommended way to set up a development environment is to turn your checkout into a virtualenv and then install the package in editable form as follows:

```
$ virtualenv .
$ bin/pip install -U -e .[test,build]
```

4.2 Running the tests

Once you've set up a virtualenv, the tests can be run as follows:

```
$ bin/nosetests
```

4.3 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ source bin/activate
$ cd docs
$ make html
```

To check that the description that will be used on PyPI renders properly, do the following:

```
$ python setup.py --long-description | rst2html.py > desc.html
```

The resulting `desc.html` should be checked by opening in a browser.

To check that the README that will be used on GitHub renders properly, do the following:

```
$ cat README.rst | rst2html.py > readme.html
```

The resulting `readme.html` should be checked by opening in a browser.

4.4 Making a release

To make a release, just update the version in `setup.py`, update the change log, tag it and push to https://github.com/cjw296/chide and Travis CI should take care of the rest.

Once Travis CI is done, make sure to go to https://readthedocs.org/projects/chide/versions/ and make sure the new release is marked as an Active Version.

CHAPTER 5

Changes

5.1 2.1.2 (31 Jan 2019)

- Fix bug in *Collection* occurring when sample attributes were not hashable.

5.2 2.1.1 (9 Jan 2019)

- Exclude tests from distribution.

5.3 2.1.0 (22 Nov 2018)

- add *attributes()*.

5.4 2.0.1 (16 Jun 2016)

- Fix nasty bug when using with `sqlalchemy` where related objects could get added to the session even though they were never requested, as a result of a backref on a third model.

5.5 2.0.0 (21 Apr 2016)

- Backwards incompatible change to split the concern for sample object identity out into *chide.Set* and `chide.sqlalchemy.Set` to avoid module-level *Collection* instances resulting in cross-test pollution.

5.6 1.0.2 (15 Apr 2016)

- Fix release faults.

5.7 1.0.0 (14 Apr 2016)

- Initial release

CHAPTER 6

License

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index