Taal Documentation

Release 0.8.2

onefinestay

December 03, 2014

Contents

Taal is a framework for translating your data. It plugs in to e.g. SQLAlchemy or Kaiso, providing a `TranslatableString` field type and a mechanism for storing and retrieving content in multiple languages.

For use-cases where the most common interaction with the translated data is for reading, an application can be set up so that language context and translations are handled centrally, after which business logic can be written almost as it would for a single-language app.

CHAPTER 1

Philosophy

Taal uses a two-phase process for managing translatable data. Upon retrieval, data is marked up as "requires translation". Subsequently (typically higher up in the stack, e.g. in some middleware), information about which particular language we are interested in may be supplied to find the actual translation string.

CHAPTER 2

Example use

```
class MyModel(Base):
    __tablename__ = "my_model"

    id = Column(Integer, primary_key=True)
    name = Column(TranslatableString())

>>> instance = session.query(MyModel).first()
>>> instance.name
<TranslatableString: (...)>

>>> translator = get_translator('en')
>>> translator.translate(instance.name)
"Spam"
```

CHAPTER 3

Development

To make your life easier, create a `setup.cfg` file with a `[pytest]` section to define your database and neo4j connection strings:

```
$ cat setup.cfg
[pytest]
addopts= --neo4j_uri=http://... --db_uri=mysql://...
```

(Note that pytest gets upset if you indent the `addopts` line)

CHAPTER 4

Contents

4.1 Quickstart

4.1.1 SQLAlchemy

Create your models, using the column type `TranslatableString`:

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

from taal import Translator
from taal.sqlalchemy import TranslatableString, events


Base = declarative_base()


class Translation(TranslationMixin, Base):
    __tablename__ = "test_translations"


class MyModel(Base):
    __tablename__ = "my_model"

    id = Column(Integer, primary_key=True)
    identifier = Column(String(20))
    name = Column(TranslatableString())  # will be translated
```

Attributes are automatically (and immediately) converted to placeholders:

```
>>> instance = MyModel(id=1)
>>> instance.name
<TranslatableString: (taal:sa_field:my_model:name, [1], None)>
```

Register your session to have translations automatically persisted:

```
>>> translator_session = Session()  # for use by the translator
>>> session = Sesssion()
>>> translator = Translator(Translation, translator_session, 'en')
translator.bind(session)  # register your session for translations

>>> instance.name = "Spam"
>>> session.add(instance)
```

```
>>> session.commit()

# the translated value is automatiaclly inserted into the translations
# table, along with some contextual information
>>> translation = session.query(Translation).first()
>>> translation.context, translation.message_id, translation.language,
... translation.pending_value
('taal:sa_field:my_model:name', '[1]', 'en', 'Spam')

>>> instance.name
<TranslatableString: (taal:sa_field:my_model:name, [1], None)>

>>> translator.translate(instance.name)
'Spam'
```

You can also translate a `TranslatableString` inside a python data structure:

```
>>> translator.translate({'key': ['list', instance.name]})
{'key': ['list', 'Spam']}
```

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*