Консервация объектов в Python

Мазаев Павел

7 ноября 2016 г.

Что мы можем иметь в виду под этим?

- Сохранить объекты языка Python для дальнейшей работы с ними
- Сохранить данные для работы с ними другим образом

Модули для консервации объектов Python

- Marshall
- Pickle & cPickle
- Shelve

Модуль marshal

- Им не надо пользоваться, но про него стоит знать
- Предназначается для внутренних нужд интерпретатора
- Не подсчитывает ссылки на один и тот же объект
- Нельзя использовать для собственных классов
- Не гарантируется совместимость между версиями

Модуль `pickle`

- Основной способ хранения объектов в Python
- Умная работа ссылками
- Возможно использование для практически произвольных классов
- Стандартный модуль, если есть необходимость, можно работать с этим форматом при помощи `C++ Boost`

Протоколы `pickle`

- Версия 0 – `ASCII` протокол. По умолчанию используется во второй версии языка.
- Версия 1. Старый бинарный протокол
- Версия 2. Бинарный протокол. Эффективное хранение классов нового типа
- Версия 3. Эффективный бинарный протокол. По умолчаию используется в 3-й версии.
- Версия 4. Появился в Python 3.4

Совместимость

- В версии 0 могут наблюдаться проблемы с кроссплатформенностью
- Бинарные форматы кроссплатформенны
- Гарантируется обратная совместимость
- Во второй версии языка стандартный модуль не поддерживает протоколы выше 2-го

cPickle

- Быстрая реализация модуля `pickle` на `C` (утверждается, что вплоть до 1000 раз)
- Формат данных совместим с `pickle`
- Присутствует по умолчанию
- Актуален только для второй версии языка: в третьей `pickle` уже сам реализован на `C`
- Есть небольшие отличия в API от стандартного `pickle`, но в большинстве задач это не проявляется

Все эти модули ***не гарантируют безопасности*** *от ошибочных или вредоносных данных.*

Не расконсервируйте данные *не из доверенного источника.*

`https://blog.nelhage.com/2011/03/exploiting-pickle/`

Использование модулей. Общий интерфейс

```
1 import serialization_module as sz
2 sz.dump(value, file_object, version_or_protocol)
3 value = sz.load(file_object)
4 representation = sz.dumps(value,
   ↪  version_or_protocol)
5 value = sz.loads(representation)
```

Модуль Pickle в Python 2

```
1  import pickle
2  with open("data.pickle", "w") as out_file:
3          pickle.dump(value, out_file, protocol=0)
4          #Осторожно! Не используйте Ctrl+C, Ctrl+V
5          #при открытии файла на чтение!
6          #Когда-нибудь вы ОБЯЗАТЕЛЬНО
7          #забудете поменять "w" на "r"
8  with open("data.pickle", "r") as in_file:
9          value = pickle.load(in_file)
10         ascii_string = pickle.dumps(value,
    ↪  protocol=0)
11         value = pickle.loads(ascii_string)
```

Модуль Pickle в Python 2

```
1  #pickle.HIGHEST_PROTOCOL == 2
2  #"w" стало "wb"
3  with open("data.pickle", "wb") as out_file:
4          #Используем протокол 2
5          pickle.dump(value, out_file,
    ↪  protocol=pickle.HIGHEST_PROTOCOL)
6  with open("data.pickle", "rb") as in_file:
7          value = pickle.load(in_file)
8          ascii_string = pickle.dumps(value,
    ↪  protocol=pickle.HIGHEST_PROTOCOL)
9          value = pickle.loads(ascii_string)
```

Модуль Pickle в Python 3

```
1  #pickle.HIGHEST_PROTOCOL == 4,
2  #но по умолчанию используется protocol = 3
3  with open("data.pickle", "wb") as out_file:
4          pickle.dump(value, out_file)
5  with open("data.pickle", "rb") as in_file:
6          value = pickle.load(in_file)
7          #В Python 3 строка и строка байтов - разные
    ↪  типы
8          byte_array = pickle.dumps(value)
9          value = pickle.loads(byte_array)
```

Что можно хранить при помощи модуля `pickle`

- `None`, `True`, `False`
- `int`, `float`, `complex`
- `u''`, `b''`, `''`
- `tuple`, `list`, `set`, `dict` с хранимыми объектами
- функции, определённые на верхнем уровне имён модуля
- классы, определённые на верхнем уровне имён модуля
- объекты таких классов, для которых `__dict__` или результат `__getstate__()` можно сохранить

- То есть нельзя консервировать лямбды, файлы, сетевые соединения и так далее.
- Если логика требует, чтобы объект был пригоден для хранения, можно реализовать собственные методы `__getstate__` и `__setstate__`
- Расконсервируемый класс должен быть доступен в модуле, где происходит расконсервация!
- То же самое с функциями!
- `pickle` умеет работать с рекурсивными данным, но если глубина рекурсии слишком

Модуль `dill`

- Не часть стандартной библиотеки
- Поддерживается и развивается
- Имеет такой же интерфейс, как и `pickle`
- Позволяет хранить некоторые дополнительные типы данных, например лямбды, срезы, код, модули.
- Позволяет сохранить сессию интерпретатора одной командой `dill.dump_session(filename)`, загрузить `dill.load_session(filename)`
- Позволяет интерактивно исследовать проблемы с консервацией (в том числе для обычного `pickle`)

Модуль shelve

Персистентный словарь, способный хранить те же типы объектов, что и pickle

```
1  import shelve
2  #В версии 2.7 with-выражение не поддерживается.
3  #Можно воспользоваться contextlib.contextmanager
4  #В python 2
5  # d = shelve.open(filename, writeback=False)
6  with shelve.open(filename, writeback=False) as d:
7          d[key] = data
8          data = d[key]
9          del d[key]
10         flag = key in d
11         #медленная операция
12         klist = list(d.keys())
```

Модуль shelve (продолжение)

```
1           #d был открыт с флагом writeback=False
2           # Всё в порядке
3           d['xx'] = [0, 1, 2]
4           # d['xx'] == [0, 1, 2]
5           d['xx'].append(3)
6
7           #надо так
8           temp = d['xx']
9           temp.append(5)
10          d['xx'] = temp
11          #или writeback=True, что замедлит закрытие
    ↪   базы данных
12          #и будет потреблять много памяти
13  #В python 2
14  #d.close()
```

Внимание!

Модуль `shelve` использует соответствующий модуль `pickle`, так что испытывает такие же проблемы с протоколами!

Сохранение словарей в JSON

```
1 import json
2 json_string = '{"first_name": "Guido",
   ↪  "last_name":"Rossum"}'
3 #yields a dict
4 parsed_json = json.loads(json_string)
5 new_json_string = json.dumps({"a":10, "b":1})
```

Можем хранить unicode, int, float, NoneType, bool, list, dict.

Источники

- http://docs.python-guide.org/en/latest/
- https://docs.python.org/3.5/library/pickle.html
- https://docs.python.org/3.5/library/shelve.html
- https://docs.python.org/3.5/library/marshal.html
- https://github.com/uqfoundation/dill