Модуль log2d в Python

Пр… …' модуля logging в Python

Есл… …, что предлагает отличный встроенный модуль logging, то можно потратить часы (или дни), раб… …тацией или производными учебными пособиями, такими как видео и статьями. Чтобы сделать что-то … …одномерное ведение журнала (выбирая из уровней DEBUG, INFO, ERROR и т.д.), необходимо раз… …включает средства форматирования, обработчики потоков, обработчики файлов и обработчики чер… …воения материала приходит понимание, что в будущем придется копировать большие куски шаб… …арий.

Есл… …дение журнала logging в какой-то существующий проект или стоит цель - просто начать работу с про… …ами, то сторонний модуль log2d уже делает все это практически в несколько строк. Вот где хор… …на вес золота. Другими словами log2d - простой инструмент для занятых людей.

Чт… …2d?

- …Можно отправлять вывод на консоль stdout или в файл, или и то, и другое.
- Можно выбрать из готовых шаблонов: форматирование сообщений и форматирование даты/времени или указать свой собственный.
- Можно выбрать между ведением журнала в один постоянно растущий файл журнала или созданием нового файла для каждого сеанса.
- Можно настроить автоматическое чередование файлов резервных копий журнала.
- Можно создавать отдельные (и/или общие) журналы для конкретных модулей, классов и объектов.
- Можно создавать быстрые сообщения журнала (используя уровень ведения журнала по умолчанию) с помощью удобных ссылок быстрого доступа.

Установка log2d в виртуальное окружение.

Так как модуль log2d не входит в стандартную библиотеку Python, его необходимо установить отдельно. Сделать это можно с помощью менеджера пакетов pip.

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# обновляем `pip`
(VirtualEnv):~$ python3 -m pip install -U pip
# ставим модуль `log2d`
(VirtualEnv):~$ python3 -m pip install -U log2d
```

Основное использование модуля log2d.

Именованный регистратор.

Код создает именованный регистратор, который выводит сообщения только на консоль stdout, используя форматирование сообщения и формат даты по умолчанию:

```
>>> from log2d import Log
>>> Log("main")
# <log2d.Log object at 0x7f469e63ccd0>
>>> Log.main.critical("Опаньки!")
# main|CRITICAL|2022-11-28T12:29:21+0300|Опаньки!
>>> Log.main.warning("Опасность!")
# main|WARNING |2022-11-28T12:29:47+0300|Опасность!
```

Вместо методов .critical .warning можно использовать любой из [стандартных уровней журнала модуля logging](), в верхнем или нижнем регистре: DEBUG, INFO, WARNING, ERROR и CRITICAL.

Регистратор, который выводит сообщения в файл.


```
>>> log_success = Log("success", to_file=True)
>>> log_success("log2d for the win!")
>>> Log.success.critical("Alert! Alert!")
# (Создает и обновляет ./success.log)
# success|DEBUG   |2022-11-28T12:36:02+0300|log2d for the win!
# success|CRITICAL|2022-11-28T12:36:06+0300|Alert! Alert!
```

"*Разумная*" логика log2d по умолчанию предполагает, что если указать ТОЛЬКО аргумент to_file, то это значит нужно, чтобы вывод направлялся ТОЛЬКО в файл, а вывод на консоль (аргумент to_stdout) автоматически устанавливается в False.

Значения по умолчанию для to_stdout и to_file равны True и False соответственно. Другими словами, если не указан ни один из аргументов, то журнал будет вестись только на консоли.

Режим ведения журнала файлов по умолчанию - добавление сообщений к существующему файлу журнала на неопределенный срок. Ниже рассматривается режим ведения журнала файла w, который перезаписывает файл журнала каждый раз при перезапуске скрипта.

Папка/директория для конкретного регистратора.

```
Log("my_title", path="./output")
```

- Аргумент to_file автоматически устанавливается в True, если указан путь path.
- Путь по умолчанию - текущий рабочий каталог или '.'.
- Если указана несуществующая папка/каталог, то будет вызвана ошибка FileNotFoundError.

Минимальный уровень сообщений.

Минимальный уровень сообщения для захвата устанавливается на английском языке, без использования числовых значений:

```
Log("my_title", level="WARNING")
```

Ссылка для регистрации сообщений на уровне по умолчанию.

Для простых скриптов без импорта или с небольшим количеством импорта можно создать свои собственные ссылки быстрого доступа, например:

```
# создаем ссылку log_failure
>>> log_failure = Log("failures")
# регистрируем сообщение
>>> log_failure("Здесь устанавливается сообщение об ошибке")
# failures|DEBUG   |2022-10-25T19:35:06+0100|Здесь устанавливается сообщение об ошибке
```

Здесь применяются обычные соображения относительно пространств имен, и для более длинных/сложных сценариев может быть разумнее придерживаться явного соглашения об именах Log.logger_name.

Ссылки быстрого доступа можно использовать, например, чтобы перезаписать [функции print()]() в существующем коде и преобразовать каждую старую строку с print() в команду регистрации. Например:

```
# пишем в начале скрипта/модуля
# 1 - переименовываем функцию `print` в `_print`
_print = print;
# 2 - присваиваем `print` значение регистратора
print = Log("print")
```

> ***Внимание!*** *Уровень журнала по умолчанию, используемый log2d - DEBUG, тогда как уровень по умолчанию модуля logging - WARNING. Это изменение призвано сделать вещи более безопасными и предсказуемыми для новых пользователей, которые в противном случае могли бы отправлять сообщения уровня DEBUG и INFO и задаваться вопросом, почему они не регистрируются.*

Имена регистраторов.

1. Можно создать имя регистратора с пробелами и другими символами, а не только с символами подчеркивания, но тогда я будет использовать удобную точечную нотацию Python object.attribute. Если бы имя журнала было, например, *main*

log', то нужно было бы вместо точечной нотации использовать getattr(Log, 'main log').warning('...'), что немного запутанно. Лучше просто использовать символы подчеркивания.

2. Как и в стандартном модуле logging, имя 'root' зарезервировано для специального типа регистратора, который фактически наследуется от других регистраторов. Это может быть очень полезно, если нужен один 'master' регистратор, который записывает абсолютно все (что немного раздражает).

Пример, чтобы продемонстрировать, как это работает:

```
>>> Log("main")
>>> Log.main.info("Это главный регистратор")
# main|INFO   |2022-10-08T23:47:08+0100|Это главный регистратор
>>> Log("root", fmt=Log.presets['timestamp_only'])
>>> Log.root.info("Это сообщение предназначено только для ROOT")
# 2022-10-08T23:52:09+0100|Это сообщение предназначено только для ROOT
Log.main.info("Это сообщение будет повторено ОБОИМИ регистраторами")
# main|INFO   |2022-10-08T23:49:23+0100|Это сообщение будет повторено ОБОИМИ регистраторами
# 2022-10-08T23:49:23+0100|Это сообщение будет повторено ОБОИМИ регистраторами
```

Если не надо использовать эту функцию "*наследования*", тогда просто необходимо избегать использования имени 'root' для любого из регистраторов log2d.

Варианты ключевых слов и полезные методы.

Новый файл журнала для каждого сеанса.

Можно настроить создание нового файла журнала для каждого сеанса/запуска, который будет перезаписывать предыдущий файл (удобно при отладке):

```
results = Log("session_results", to_file=True, mode="w")
```

Сменяющие друг друга резервные копии журнала

Можно настроить создание нового файла журнала для каждого сеанса и автоматически создать 10 сменяющих друг друга резервных копий:

```
results = Log("session_results", to_file=True, mode="w", backup_count=10)
```

Текущим файлом журнала всегда будет session_results.log, но для последующих сеансов он будет скопирован в session_results.log.1, затем в session_results.log.2 и т.д. до тех пор, пока не будет достигнуто количество резервных копий backup_count, а затем запустится снова на ротационной основе.

Если backup_count не указан, то количество резервных копий по умолчанию равно 5.

Определение формата сообщения и/или формата даты.

Предварительный просмотр определенного формата сообщения и/или формата даты - либо одного из поставляемых шаблонов, либо собственного дизайна:

```
Log.preview(fmt=Log.presets["timestamp_only"], datefmt=Log.date_formats["time"])
# 13:10:06|This is a preview log entry.
Log.preview(datefmt="%m-%d::%H:%M")
# temp_preview|09-25::15:36|This is a preview log entry.
```

Просмотр всех комбинаций шаблонов сообщений/дат:

```
>>> Log.preview_all()
# Вывод длинный, вот некоторые из них
# ...
# fmt="name_and_time", datefmt="am_pm"
# temp_preview|28/11/2022 01:57:10 PM|This is a preview log entry.
# ...
# fmt="timestamp_only", datefmt="iso8601"
# 2022-11-28T13:57:10+0300|This is a preview log entry.
# ...
# fmt="file_func_name", datefmt="date_and_time"
# WARNING |2022-11-28 13:57:10|line 100 of function: preview in __init__.py|This is a preview log entry.
# ...
# le_func_name", datefmt="iso8601"
#  |2022-11-28T13:57:10+0300|line 100 of function: preview in __init__.py|This is a preview log entry.
```

```
# ...
# fmt="relative_time", datefmt="iso8601"
# WARNING |5277620|~/.venv/lib/python3.8/site-packages/log2d/__init__.py|preview|100|This is a preview log entry.
```

Как использовать?

Предустановленные шаблоны:

```
>>> from log2d import Log
>>> datefmt = Log.date_formats["date_and_time"]
>>> fmt = Log.presets["file_func_name"]
>>> log = Log("main", fmt=fmt, datefmt=datefmt)
>>> log.main.debug("Тест")
# DEBUG   |2022-11-28 14:32:21|line 1 of function: <module> in <stdin>|Тест
```

Составление собственных форматов:

```
>>> from log2d import Log
>>> fmt = '%(asctime)s:%(levelname)s:%(name)s:%(message)s'
>>> datefmt = '%d/%m/%Y %I:%M:%S %p'
>>> log = Log("main1", fmt=fmt, datefmt=datefmt)
>>> log.main1.warn("Тест")
# 28/11/2022 02:40:09 PM:WARNING:main1:Тест
```

> *Более подробно о составлении собственных форматов смотрите в материалах:*
>
> - *"Класс Formatter() модуля logging в Python".*
> - *"Объект LogRecord() модуля logging в Python".*

Добавление нового формата даты или сообщения.

Добавим новый формат даты или формат сообщения на уровне класса, чтобы будущие экземпляры могли их использовать:

```
>>> from log2d import Log
>>> fmt = "%(asctime)s (%(name)s): %(message)s"
>>> datefmt = "%m-%d %H:%M"
>>> Log.presets["my_message_format"] = fmt
>>> Log.date_formats["my_date_format"] = datefmt
```

Список всех экземпляров журнала:

```
>>> Log.index
# {
#   'main': <log2d.Log object at 0x7f469e63ccd0>,
#   'success': <log2d.Log object at 0x7f469e5b6640>
# }
```

Доступ к базовому объекту logging.Logger.

Для еще большего контроля можно получить доступ к базовому объекту logging.Logger:

```
>>> log = Log("main").logger
>>> type(log)
# <class 'logging.Logger'>
>>> dir(log)
# [...
#   'addFilter', 'addHandler', 'callHandlers', 'critical',
# ...]
```

Рецепты/примеры использования модуля log2d.

Один файл журнала для каждого модуля.

```
# файл `test.py`
from log2d import Log, Path

if __name__ == '__main__':
    log = Log(Path(__file__).stem, to_file=True).logger

    # Затем просто повторно используйте объект
    # журнала в другом месте скрипта, например:
```

```
    file_name = Path(__file__).name
    log.critical(f'critical message from: {file_name}')
    log.error(f'error message from: {file_name}')
    log.warning(f'new warning message from: {file_name}')
    log.info(f'info message from: {file_name}')
    log.debug(f'debug message from: {file_name}')

# файл `test.log`
# my_file|CRITICAL|2022-11-28T14:32:50+0100|critical message from: test.py
# my_file|ERROR   |2022-11-28T14:32:50+0100|error message from: test.py
# my_file|WARNING |2022-11-28T14:32:50+0100|new warning message from: test.py
# my_file|INFO    |2022-11-28T14:32:50+0100|info message from: test.py
# my_file|DEBUG   |2022-11-28T14:32:50+0100|debug message from: test.py
```


Собственная запись в журнале для каждого экземпляра класса.

```
from log2d import Log

class MyClass:
    def __init__(self, name):
        params = {"fmt": Log.presets["name_and_time"]}
        self.log = Log.index.get(name) or Log(name, **params)

    def method_1(self):
        # Do something
        self.log("method_1 did something!")

x = MyClass("Instance 1")
x.method_1()
y = MyClass("Instance 2")
y.method_1()
x.log("This message was logged directly")
y.log("Likewise, but different instance")

# Instance 1|2022-11-28T14:15:01+0300|method_1 did something!
# Instance 2|2022-11-28T14:15:01+0300|method_1 did something!
# Instance 1|2022-11-28T14:15:01+0300|This message was logged directly
# Instance 2|2022-11-28T14:15:01+0300|Likewise, but different instance
```

Собственная запись в журнале для каждого класса и его экземпляров.

```
from log2d import Log

class MyAbstractClass:
    def __init__(self, name, *args, **kwargs):
        params = {"fmt": Log.presets["name_and_time"]}
        self.log = Log.index.get(name) or Log(name, **params)

class MyClass(MyAbstractClass):
    def __init__(self, name, *args, **kwargs):
        super().__init__(self.__class__.__name__, *args, **kwargs)
        self.name = name

    def method_1(self):
        # Do something
        self.log(f"method_1 of {self.name} did something!")

x = MyClass("Instance X")
x.method_1()
y = MyClass("Instance Y")
y.method_1()
x.log(f"This message was logged by {x.name}")
y.log(f"And this one by {y.name}")

# MyClass|2022-11-28T14:22:59+0300|method_1 of Instance X did something!
# MyClass|2022-11-28T14:22:59+0300|method_1 of Instance Y did something!
# MyClass|2022-11-28T14:22:59+0300|This message was logged by Instance X
# MyClass|2022-11-28T14:22:59+0300|And this one by Instance Y
```

Вверх

Пример настройки логирования.

```
from log2d import Log

ROOT_DIR = "./my_app_path"

Log.path = f"{ROOT_DIR}/logs/"
Log.fmt = Log.presets["name_and_time"]
Log.datefmt = Log.date_formats["time"]
Log.to_file = True
Log.to_stdout = False
Log.mode = "w"
Log.backup_count = 10

>>> Log("main", to_stdout=True)
# <log2d.Log object at 0x7fe75dc50b20>
>>> Log("selenium")
# <log2d.Log object at 0x7fe75dcdac40>
>>> Log("timings")
# <log2d.Log object at 0x7fe75d955400>
>>> Log("results")
# <log2d.Log object at 0x7fe75d955580>
>>> Log("retries")
# <log2d.Log object at 0x7fe75d955700>
>>> Log("errors")
# <log2d.Log object at 0x7fe75d955880>
```


Вверх