skypro

Библиотека logging. Шпаргалка

Логирование

Логирование в Python — это процесс записи информации о том, что происходит в программе во время ее работы.

```
import logging

name = "Alice"
age = 30

# Выводим сообщение в лог
logging.info (f"Имя: {name}, возраст: {age}")
```

Уровни логирования

1. `DEBUG` — сообщения для отладки приложения.
2. `INFO` — информационные сообщения.
3. `WARNING` — предупреждения.
4. `ERROR` — сообщения об ошибках.
5. `CRITICAL` — критические сообщения.

Для логирования сообщений различного уровня в библиотеке `logging` предусмотрены соответствующие методы:

```
app_logger.debug("Это сообщение уровня DEBUG")
app_logger.info("Это сообщение уровня INFO")
app_logger.warning("Это сообщение уровня WARNING")
app_logger.error("Это сообщение уровня ERROR")
app_logger.critical("Это сообщение уровня CRITICAL")
```

Эти методы добавляют сообщения в лог на соответствующем уровне:

```
2023-10-17 14:52:29,394 DEBUG: Это сообщение уровня DEBUG
2023-10-17 14:52:29,394 INFO: Это сообщение уровня INFO
2023-10-17 14:52:29,394 WARNING: Это сообщение уровня WARNING
2023-10-17 14:52:29,394 ERROR: Это сообщение уровня ERROR
2023-10-17 14:52:29,394 CRITICAL: Это сообщение уровня CRITICAL
```

Логирование на различных уровнях в программе:

```
# Импортируем модуль logging
import logging

# Получаем корневой логер
logger = logging.getLogger()

# Логируем сообщение уровня ERROR
logger.error("Это ошибка")

# Получаем логер с определенным именем
named_logger = logging.getLogger("mylogger")

# Логируем сообщение уровня CRITICAL
named_logger.critical("Очень критично")
```

Компоненты логирования

1. `Logger` — это объект, который пишет логи. Он принимает сообщения от программы и создает для них записи лога. Он также знает, какого уровня должны быть логи.

   > Логер нужен для того, чтобы писать логи.

2. `Handler` — это объект, который отправляет логи. Он принимает записи лога от логера и решает, куда их отправить. Например, он может отправить логи в файл на компьютере или на экран консоли.

   > Хендлер нужен для того, чтобы выбирать место для логов.

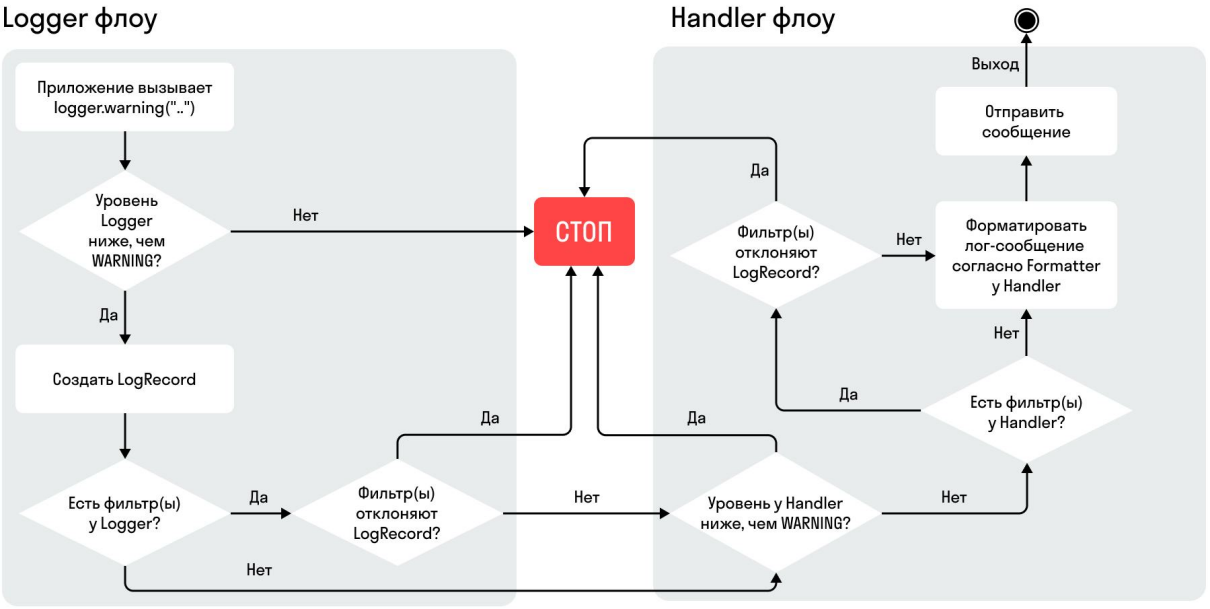
3. `Filter` — это объект, который фильтрует логи. Он принимает записи лога от логера или хендлера и решает, какие из них нужно обработать, а какие нет. Например, он может пропускать логи ниже определенного уровня или по определенному слову.

   > Фильтр нужен для того, чтобы отсеивать лишние логи.

4. `Formatter` — это объект, который форматирует логи. Он принимает записи лога от хендлера и решает, как они должны выглядеть. Например, он может добавить дату, время, уровень и другие данные к логам.

   > Форматер нужен для того, чтобы делать логи красивыми и понятными.

`LogRecord` — это объект, созданный `Logger`, который содержит всю информацию о сообщении лога.



Логеры

`Logger` — это объект, предназначенный для записи логов. Он получает сообщения от приложения, создает для каждого сообщения объект `LogRecord`, назначает ему уровень важности (`DEBUG`, `INFO`, `WARNING` и т. д.) и передает его в обработчики (`Handlers`).

Создание и получение логеров

```python
import logging

# Получение корневого логера
root_logger = logging.getLogger()

# Создание и получение именованного логера
app_logger = logging.getLogger("my_application")
```

В качестве аргумента для функции `getLogger()` можно использовать параметр `__name__`, который установит имя логера равное имени модуля.

```
import logging

# Получение корневого логера
root_logger = logging.getLogger()

# Создание и получение именованного логера
app_logger = logging.getLogger(__name__)
```

Установка уровня логирования

Логер позволяет установить уровень важности сообщений, которые будут записываться. Это делается с помощью метода `setLevel()`:

```
import logging

app_logger = logging.getLogger("my_application")
app_logger.setLevel(logging.DEBUG)
```

Хендлеры

`Handler` **(обработчик)** — это объекты, которые определяют, куда и как выводить логи.

StreamHandler

`StreamHandler` — тип хендлера, который используется для вывода логов в консоль.

```
import logging

logger = logging.getLogger(__name__)
# Создаем хендлер для вывода в консоль
console_handler = logging.StreamHandler()
logger.addHandler(console_handler)
logger.setLevel(logging.DEBUG)

logger.debug('Debug message')
```

FileHandler

`FileHandler` — тип хендлера, который используется для вывода логов в файл.

```
import logging

logger = logging.getLogger(__name__)
# Создаем хендлер для вывода в файл
file_handler = logging.FileHandler('example.log')
logger.addHandler(file_handler)
logger.setLevel(logging.DEBUG)

logger.debug('Debug message')
```

Форматеры

`Formatter` — это тот компонент библиотеки `logging`, который определяет, как выводить сообщения логов. Он связан только с обработчиком и не влияет на логирование сообщений.

```
import logging

logger = logging.getLogger(__name__)
file_handler = logging.FileHandler('example.log')
file_formatter = logging.Formatter('%(asctime)s %(levelname)s: %(message)s')
file_handler.setFormatter(file_formatter)
logger.addHandler(file_handler)
logger.setLevel(logging.DEBUG)

logger.debug('Debug message')
logger.info('Info message')
logger.warning('Warning message')
logger.error('Error message')
logger.critical('Critical message')
```

Когда мы запустим код, то увидим, что файл `example.log` был создан в рабочей директории и содержит подобный текст:

```
2023-10-17 14:52:29,394 DEBUG: Debug message
2023-10-17 14:52:29,394 INFO: Info message
2023-10-17 14:52:29,394 WARNING: Warning message
2023-10-17 14:52:29,394 ERROR: Error message
2023-10-17 14:52:29,394 CRITICAL: Critical message
```

Полный список параметров для форматирования:

- `%(asctime)s` — дата и время события логирования.
- `%(name)s` — имя логера.
- `%(levelname)s` — уровень логирования.
- `%(message)s` — текст сообщения.
- `%(filename)s` — имя файла, в котором произошло событие.
- `%(funcName)s` — имя функции, в которой произошло событие.
- `%(lineno)d` — номер строки, в которой произошло событие.
- `%(process)d` — ID процесса.
- `%(thread)d` — ID потока.

Пример логирования в реальной программе

```
import logging

# Основная конфигурация logging
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s - %(name)s - %(levelname)s - %(mess
age)s',
                    filename='application.log',  # Запись логов в файл
                    filemode='w')  # Перезапись файла при каждом запуске

# Создаем логеры для различных компонентов программы
auth_logger = logging.getLogger('app.auth')
db_logger = logging.getLogger('app.database')
main_logger = logging.getLogger('app.main')
```

Функция авторизации пользователя:

```
def login(username, password):
    # Записываем информацию о попытке входа в лог
    auth_logger.info(f'Попытка входа для пользователя: {username}')

    # Проверяем правильность введенного имени пользователя и пароля
    if username == "admin" and password == "secret":
        # Если проверка успешна, записываем сообщение
        # об успешной авторизации
        auth_logger.info('Успешная авторизация')
        return True
    else:
        # Если проверка неуспешна, записываем предупреждение
        # о неудачной попытке входа
        auth_logger.warning('Неудачная попытка входа')
        return False
```

Функции работы с базой данных:

```
data = {}  # Используем словарь как простую базу данных

def insert(key, value):
    # Записываем информацию о вставке данных
    db_logger.info(f'Вставка данных: {key} = {value}')
    # Вставляем данные в словарь
    data[key] = value

def select(key):
    # Пытаемся получить значение по ключу из словаря
    value = data.get(key)
    # Проверяем, найдено ли значение
    if value:
        # Если значение найдено, записываем информацию
        # об успешном получении данных
        db_logger.info(f'Получены данные: {key} = {value}')
    else:
        # Если значение не найдено, записываем предупреждение
        # о ненайденных данных
        db_logger.warning(f'Данные с ключом {key} не найдены')
    return value
```

Основная функция:

```
def main():
    try:
        # Записываем сообщение о запуске приложения
        main_logger.info('Запуск приложения')

        # Попытка авторизации
        user_logged_in = login("admin", "secret")

        if user_logged_in:
            # Если авторизация успешна, работаем с базой данных
            insert("user_id", "12345")
            select("user_id")
            select("non_existing_id")
        else:
            # Если авторизация не успешна, записываем предупреждение
            # и прекращаем работу
            main_logger.warning("Неудачная авторизация. Прекращение работы")

    except Exception as e:
        # Записываем ошибку, если произошло исключение
        # во время выполнения программы
        main_logger.error(f'Произошла ошибка: {e}', exc_info=True)

    finally:
        # Записываем сообщение о завершении работы приложения
        main_logger.info('Завершение работы приложения')

if __name__ == "__main__":
    main()
```

Что будет в логах:

```
2023-10-17 15:35:23,456 - app.main - INFO - Запуск приложения
2023-10-17 15:35:23,457 - app.auth - INFO - Попытка входа для пользователя:
admin
2023-10-17 15:35:23,457 - app.auth - INFO - Успешная авторизация
2023-10-17 15:35:23,458 - app.database - INFO - Вставка данных: user_id =
12345
2023-10-17 15:35:23,459 - app.database - INFO - Получены данные: user_id =
12345
2023-10-17 15:35:23,459 - app.database - WARNING - Данные с ключом non_exis
ting_id не найдены
2023-10-17 15:35:23,460 - app.main - INFO - Завершение работы приложения
```