# Урок 1. Логирование со structlog и loguru

На первом занятии давайте поговорим про альтернативные инструменты логирования — structured logging и loguru. Для чего они могут быть полезны и какие возможности есть у этих логгеров, которых нет в стандартом логгере?

Давайте начнем со structured logging.

Ключевое слово здесь — structured, структурированные. Обычный журнал логов не структурирован. Представленные в текстовом виде события лога в целом легко читаются разработчиком, можно найти ошибку по времени и посмотреть traceback. Но в случае, когда мы захотим парсить логи для какой-либо аналитики, разбивать сообщения по категориям, фильтровать и индексировать, могут возникнуть сложности. Ведь информация может быть неоднородна, логи разных источников могут записывать события по-разному. Для решения такой задачи отлично подойдут структурированные логи.

Структурированное ведение журнала позволяет обрабатывать сообщения логов не как текст, а как наборы данных. Идея структурированных логов состоит в том, чтобы взять журнал лога, который фиксирует события в виде строки текста, и преобразовать его в набор данных в формате «ключ/значение», в котором будет проще искать и анализировать.

Для python основной библиотекой для структурированных логов является библиотека structlog. Основными достоинствами данной библиотеки являются:

* лёгкое ведение журнала, где каждая строка представляет набор пар «ключ/значение»;
* постепенное построение контекста события: работая с логом, как со словарём, мы можем постепенно добавлять ключи в событие;
* наличие процессоров, которые выполняют настраиваемые действия над объектом события и возвращают новый объект для записи в журнал. Так мы можем дописывать, удалять или фильтровать события;
* гибкое форматирование, которое позволяет гибко настраивать создание записей в журнал. Мы можем раскрасить наши логи, хранить их в json-формате или использовать какой-то стандартный формат. Предобработка создания записей также выполняется с помощью встроенных процессоров;
* совместимость со стандартным логгером: описывая конфиг базового логгера, мы может использовать его в structlog.

| **from flask import Flask, request, g**  **import structlog**  **from structlog.stdlib import LoggerFactory**  **import logging**  **import sys**  **logging.basicConfig(format="%(message)s",**  **stream=sys.stdout,**  **level=logging.INFO)**  **app = Flask(\_\_name\_\_)**  **structured\_log = structlog.get\_logger()**  **import datetime**  **def timestamper(\_, \_\_, event\_dict):**  **event\_dict["time"] = datetime.datetime.now().isoformat()**  **return event\_dict**  **structlog.configure(**  **processors=[timestamper, structlog.processors.JSONRenderer()],**  **logger\_factory=LoggerFactory())**  **@app.before\_request**  **def before\_request():**  **method = request.method**  **user\_agent = request.user\_agent**  **log = structured\_log.bind(method=method, user\_agent=user\_agent)**  **g.log = log**  **@app.route('/one')**  **def one():**  **g.log.msg('route one')**  **return 'one', 200**  **@app.route('/two')**  **def two():**  **g.log.msg('route two')**  **return 'two', 200**  **if \_\_name\_\_ == '\_\_main\_\_':**  **app.run()** |
| --- |

В этом примере я описал преимущества данного логгера. Мы строим контекст события постепенно, используем процессор, выбор формата форматирования и синхронизацию с базовым логгером.

Следующий инструмент для логирования, который мы разберём, — это loguru. Ключевой причиной создания этого логгера является сложность настройки базового логгера. Более того, разработчики добавили ряд полезных функций, которые устраняют недостатки стандартных средств ведения журнала.

Основная концепция loguru состоит в том, что существует только один объект logger.

Давайте сразу разберём некоторые его преимущества:

* простота настройки: настройка логгера от loguru выглядит намного приятнее, нежели громоздкие конфиги базового логгера;
* удобное ведение лога, которое позволяет выводить в лог traceback со значениями переменных, форматировать события в JSON, отображать события цветом;
* поддержка структурированного ведения: аналогично structlog, loguru хранит контекст записи лога и позволяет постепенно добавлять ключи; также можно использовать предобработчики записи в журнал;
* поддержка базового логгера;

### упрощённое ведение журнала файлов: ротация файлов, сохранение и очистка, возможность архивирования — всё это настраивается одной строкой.

| **from flask import Flask**  **from loguru import logger**  **app = Flask(\_\_name\_\_)**  **logger.add(**  **"logs/log.log",**  **rotation='1 week',**  **compression='zip',**  **level='DEBUG',**  **format="{time} {level} {message}",**  **backtrace=True,**  **diagnose=True**  **# serialize=True**  **)**  **@app.route('/one')**  **def one():**  **logger.info('one route')**  **return 'one', 200**  **@app.route('/error')**  **def error():**  **try:**  **null\_var = 0**  **a = 1 / null\_var**  **except ZeroDivisionError:**  **logger.exception("Error")**  **return 'error'**  **if \_\_name\_\_ == '\_\_main\_\_':**  **app.run()** |
| --- |

В этом примере я описал опции записи логов в файл с ротацией раз в неделю и архивированием, отображением цепочки ошибок и значений переменной, вызвавшую ошибку. Применил декоратор logger.catch для фиксации исключений, который эквивалентен logger.exception().

Логгеры, которые мы разобрали на этом занятии, ничем не уступают базовому логированию, а в некоторых случаях на порядок превосходят его. Они отлично подходят для небольших проектов и при локальной разработке, где вам важно фиксировать ошибки и собирать статистику работы системы. Но в случае, когда речь заходит о каких-то масштабных промышленных проектах, никак не обойтись без ELK-стека. Следующие наши занятия как раз будут посвящены разбору инструментов данного стека.