Дополнительные компоненты

Signals

Отдельно можно рассмотреть *ConnectAccessor* — это наследник базового класса Accessor, но обладающий методами *connect* и *disconnect*. Сервер может вызвать метод *connect* при запуске и метод *disconnect* при завершении работы. Это позволит запустить сервер с уже готовым подключением к базе данных и освободить ресурсы после отключения. Ниже мы рассмотрим, как вызывать эти методы в нужные моменты.

Aiohttp signals позволяют выполнять какую-то логику при включении сервера, выключении и чистке ресурсов.

Пример:

```
app.on_startup.append(mongo_accessor.connect())
app.on_shutdown.append(mongo_accessor.disconnect())
```

Теперь при запуске приложения будет выполнено подключение к MongoDB, а при его завершении выполнено отключение.

Logger

Объект, который, как ни странно, умеет писать логи. В проекте обычно существует один root-логгер, от которого наследуются остальные. У logger есть ряд составных частей:

- сам logger — задает точку входа в приложение и уровень логгирования
- handler — решает, куда записать это сообщение, например — вывести в stdout, файл или отправить в Telegram-бота
- filter — пропускает к записи только часть логов, фильтруя их по заданным критериям
- formatter — определяет формат вывода логов

Каждый логгер в цепочке наследников может определять собственную реализацию каждой части.

Существует такое понятие, как уровень логгирования. Обычно есть такие уровни: DEBUG, INFO, WARNING, ERROR. Они фильтруют и не записывают в логи события, которые произошли на более низком уровне (см. пример). С помощью logging.basicConfig() можно указать общий уровень логгирования. Для прода лучше логгировать на у ошибки и предупреждения, а для локального тестирования можно поставить режим DEBUG — это предотвратит раздувание лог-файлов.

Пример создания logger'a:

```
logging.basicConfig(level=logging.INFO)
logger = getLogger()
logger.debug("не будет записано в лог")
logger.info("будет записано в лог")
```