Агрегаторы

asyncio.gather

Принимает **перечень** задач

Плюсы

- Выполняет несколько задач конкурентно
- Возвращает результаты в виде списка, где порядок результатов совпадает с порядком задач
- Не снимает другие работающие задачи в случае исключения в одной из них

Минусы

- Блокируется до конца выполнения всех задач
- Не снимает другие работающие задачи в случае исключения в одной из них
- Требуется отфильтровывать типы возвращаемых результатов на наличие ошибки

```
tasks = [task1, task2, task3]
results = await asyncio.gather(*tasks)
```

```
results = await asyncio.gather(task1, task2, task3)
```

asyncio.gather принимает *необязательный параметр*, **return_exceptions**, который позволяет указать, как мы хотим обрабатывать исключения от допускающих ожидание объектов.

Это булево значение, поэтому возможно два варианта:

`return_exceptions=False` – это режим по умолчанию.

Если хотя бы одна сопрограмма возбуждает исключение, то gather возбуждает то же исключение в точке await.

Но, даже если какаято сопрограмма откажет, остальные не снимаются и продолжат работать при условии, что мы обработаем исключение и оно не приведет к остановке цикла событий и снятию задач;

`return_exceptions=True` – в этом случае исключения возвращаются в том же списке, что результаты.

Сам по себе вызов gather не возбуждает исключений, и мы можем обработать исключения, как нам удобно.

```
results = await asyncio.gather(*tasks, return_exceptions=True)

exceptions = [res for res in results if isinstance(res, Exception)]
successful_results = [res for res in results if not isinstance(res, Exception)]
```

asyncio.as_completed

Принимает **итерируемый объект** с задачами

Плюсы

- Не требуется ждать завершения всех задач
- Возвращает итератор и помещает в него задачи, которые выполнились раньше других
- **Не снимает** другие работающие задачи в случае исключения в одной из них

Минусы

- Возвращает результаты в случайном порядке, зависящем от времени выполнения задачи
- Чтобы пройти итератор, необходимо дождаться выполнения всех задач
- **Не снимает** другие работающие задачи в случае исключения в одной из них

```
tasks = [task1, task2, task3]

for finished_task in asyncio.as_completed(tasks):
    print(await finished_task)
```

Если задача возбудит исключение, то мы сможем обработать ее сразу же, поскольку оно возникает в точке, где мы ожидаем будущего объекта с помощью await.

```
for finished_task in asyncio.as_completed(tasks):
    try:
        print(await finished_task)
    except Exception:
        print("Ошибка")
```

Функция *as_completed* предоставляет необязательный параметр **timeout** в секундах.

Он управляет временем работы as_completed; если потребуется больше, то каждый допускающий ожидание объект в итераторе возбудит исключение **TimeoutException** в точке ожидания с помощью **await**.

```
for finished_task in asyncio.as_completed(tasks, timeout=0.3):
    try:
        print(await finished_task)
    except asyncio.exceptions.TimeoutError:
        print("timeout")
```

asyncio.wait

Принимает **итерируемый объект** с задачами

Плюсы

- Можно не ждать завершения всех задач
- Возвращает управление в момен указанного события
- Можно завершить задачи, после первого исключения
- Можно завершить задачи, после первого результата

Минусы

- Необходимо дополнительно обрабатывать задачи после возврата управления

Возвращает **кортеж** из двух элементов.
Первый элемент это **множество** задач set[Task], которые **выполнились** на момент указанного события.
Второй элемент это **множество** задачset[Task], которые еще **не были выполнены** на момент указанного события.

return_when может принимать значения:

ALL_COMPLETED, **FIRST_EXCEPTION** и **FIRST_COMPLETED**, а
по умолчанию равен ALL_COMPLETED

```python
tasks = [task1, task2, task3]

done, pending = await asyncio.wait(tasks, return_when=asyncio.FIRST_COMPLETED)

for done_task in done:
    try:
        result = await done_task
        print(result)
    except Exception as e:
        print("error", e)

for pending_task in pending:
    pending_task.cancel()
```

wait не возбуждает исключения в случае тайм-аута,
в отличие от wait_for и as_completed.
Когда случается тайм-аут, wait возвращает все завершившиеся задачи,
а также те, что еще не завершились в момент таймаута.