ОС UNIX, лекция 12: асинхронная разработка

Пожидаев М. С.

28 октября 2020 г.

Неблокирующий ввод-вывод

Текущий поток исполнения инициирует операцию ввода-вывода, после чего продолжает исполнение, откладывая обработку результата на потом. Получается намного эффективнее, но появляются две проблемы:

- что выполнить для продолжения работы;
- как и когда обработать результат.

Пример: DMA

Событийная модель Node.js

1. Всё приложение разбивается на множество событий как можно меньшего размера.
2. События выполняются в одном общем потоке под управленнием главного цикла (event loop).
3. Операция ввода-вывода обычно разбивается на событие, её запускающее, и событие, обрабатывающее её результат.

Преимущества:

- можно не заботится о согласовании потоков;
- возрастает общая эффективность исполнения.

Недостатки асинхронного программирования

1. Нельзя смешивать блокирующие и неблокирующие вызовы.
2. Сильно затруднена обработка ошибок (использование исключений практически невозможно).

Лямбда-выражения

Лямбда-выражение — функция, сохраняемая и передаваемая как значение переменной. Может использоваться для объявления вспомогательной функции в произвольном месте кода. Следует тщательно следить за областью видимости переменных, с которыми ведётся работа внутри лямбда-выражения.

Три стиля неблокирующих функций

1. Использование callback-функций.
2. Использование промисов.
3. Использование `async/await`.

Пример вызова с callback-функцией

```
this.objDir = (req, res, obj)=>{
  if (this.emptyStr(obj) || obj.trim().length < 8) {
    errors.internal(req, res);
    return;
  }
  const path = obj.trim();
  fs.mkdir(path, {recursive: true}, (e)=>{
    //Some code to handle the result
  });
};
```

Промисы

Промис (promise) — заготовленный объект в JavaScript, задающий порядок отложенного выполнения кода. Может быть в одном из трёх состояний:

- pending;
- fullfilled;
- rejected.

Последующие действия задаются функцией (then()), которая может быть вызвана несколько раз (chaining).

Создание промиса

```
function fooBar() {
  return new Promise((resolve, reject)=>{
    if (ok)
      resolve ('OK'); else
      reject ('Problem');
    });
}
```

Вызов промиса

```
foobar()
.then((res)=>{})
    ...
.catch((e)=>{});
```

Промисифайзеры

Проблема промисов в том, что функция с получением результата через callback и функция, возвращающая промис, имеют два разных интерфейса. Поэтому появились так называемые промисифайзеры, которые автоматически анализируют структуру объекта и на каждую функцию с callback добавляют ещё одну, имеющую интерфейс промисифайзера. Очевидно, для работы промисифайзеры используют средства интроспекции языка.
Пример: Bluebird (добавляет функции с суффиксом `Async`).

Ключевое слово async

Ключевое слово `async` используется при объявлении функции и означает, что:

1. Функция автоматически возвращает промис.
2. Внутри функции можно использовать ключевое слово `await`.
3. Функция не может выбрасывать исключения.

Ключевое слово await

Ключевое слово `await` используется для вызова функции, возвращающей в качестве результата промис. При вызове:

1. Текущее исполнение функции прерывается, поток заполняется исполнением других событий в очереди.
2. Работа будет продолжена, когда промис перейдёт в состояние fullfilled.
3. Результат выполнения промиса будет возвращён как значение функции.

Пример асинхронной функции

```
this.objDir = async (req, res, obj)=>{
  if (this.emptyStr(obj) || obj.trim().length < 8) {
    errors.internal(req, res);
    return;
  }
  const path = obj.trim();
  await fs.mkdirAsync(path, { recursive: true });
};
```

Спасибо за внимание!

Веб-сайт: http://marigostra.ru/
E-mail: msp@luwrain.org