Задание по программированию: Параллельное выполнение асинхронных функций

Вы не отправили работу. Для успешной сдачи вам необходимо набрать 1/1 баллов.

Срок сдачи Сдайте это задание до September 16, 11:59 PM PDT

Инструкции

Моя работа

Обсуждения

Assignment: Parallel

VY

Параллельное выполнение асинхронных функций

Vladimir Yugay Assignment: Parallel · 7 months ago

А подсказки еще есть какие нибудь? Для новичка слишком сложно.

0 Лайки Reply Подписаться на обсуждение

Старые **Самые популярные** **Новые**

E Eugene Teaching Staff · 7 months ago

Владимир, с чем конкретно сложности?

0 Лайки Скрыть 11 ответы

Alexander Bukreev · 7 months ago

а как лучше решать -- с помощью callback или промисов?

0 Лайки

Alexander Bukreev · 7 months ago

2. Что должна возвращать функция next?

0 Лайки

VY Vladimir Yugay · 7 months ago

Здравствуйте, Евгений. Немного запутался в следующих моментах.

Функции переданные как аргументы выполняют асинхронный код (в checks.js), но в условный массив results значения должны попадать в порядке вызова функций (это следует из примеров в checks.js). То есть, функции надо выполнять по порядку. Однако, в случае ошибки, функция, которая первая отработала должна прервать выполнение программы. Получается противоречие. Функции (а точнее их колбеки next) должны выполняться одна за другой (то есть ждать друг друга) и в то же время асинхронно.

0 Лайки

E Eugene Teaching Staff · 7 months ago

2 Alexander Bukreev:

На мой взгляд колбеки больше подходят, но через промисы тоже получится. Функция next ничего не возвращает, её вызывают по завершению какого-то действия, результат её выполнения не важно

0 Лайки

E Eugene Teaching Staff · 7 months ago

2 Vladimir Yugay:

С точки зрения клиентского кода (check.js) нам важны следующие моменты:

- если все функции отработали успешно, в results должны быть данные в правильном порядке
- если хотябы одна из функций завершилась аварийно (неважно какая), выполнение заканчивается с этой ошибкой. Здесь нам уже не интересен results и порядок. В результирующий колбек должна прилететь первая ошибка, то есть если упала 3-я функция, а за ней 2-я, то прилетит ошибка от 3-й, а вторая просто проигнорируется.
- результирующий колбек должен вызываться ровно один раз в любой из ситуация выше.

Порядок выполнения функция не важен, это скрыто в реализации. Их можно выполнять как угодно, нужно лишь обеспечить выполнение требований.

0 Лайки

Alexander Bukreev · 7 months ago

Бился весь день.. не понимаю, как зацепить значения из setTimeout для next.

```
1  setTimeout(function () {
2                  next(null, '500ms');
3              }, 500);
```

например, 500ms или 500. Ведь функция еще не запущена. уже что только не перепробовал :((((((и через Object.create и через замедления.... Хоть намекните... прошел весь блок до 5-й неделе -- здесь тупик. решений у меня нет. в лекциях подсказок не нашел... ДОЛЖНЫ в заданиях быть подсказки, как они есть в остальных заданиях четвертого блока... в общем, полный дизапоин :(((((((

0 Лайки

E Eugene Teaching Staff · 7 months ago

Привет! я не очень понимаю, что значит "зацепить значение". SetTimeout - это просто пример асинхронной функции, этом мог быть промис, который бы выглядел так:

```
somePromise
.then(function() { next(null, 'promise!'); }))
.catch(function(err) { next(err); }))
```

Смысл в том, чтобы обработать некоторые функции, которые выполняются асинхронно, но они гарантированно вызовут переданную в них функцию next, когда все закончится, передадут в нее либо ошибку либо данные. Наша задача научиться такие функции запускать асинхронно и дожидаться их выполнения. Почти как Promise.all, только с колбеками

```
parallel([
  asyncFnOne,
  asyncFnTwo,
  asyncFnThree,
  function (err, data) {
    // сюда попадаем, когда все функции закончили свое выполнение
  }
])

// Функции примерно такие
function asyncFnOne(next) {
  // здесь мы сделаем асинхронное и вызовем next
}

function asyncFnTwo(next) {
  // здесь мы сделаем асинхронное и вызовем next
  // Например, setTimeout
  setTimeout(function () {
    // Сюда мы попадем только поистечению 500мс
    // теперь можно вызвать next с результатом
    next(null, 'привет из setTimeout')
  }, 500);
}

function asyncFnThree(next) {
  // здесь мы сделаем асинхронное и вызовем next
  // Например, прочитаем из файла
  setTimeout(function () {
    require('fs').readFile('some_file.txt', function (err, data) {
      // сюда попадем когда файл будет прочитан, теперь можно вызвать next

      next(err, data);
    });
  }, 500);
}
```

Что там в функциях абсолютно не важно. Важно лишь то, что эти функции как-то асинхронно вызовут next.

Для решения мы можем определить функцию next, которую отправить в каждую из асинхронных функций. Дальше надо только придумать, как соблюсти порядок и обработать ошибки

0 Лайки

Alexander Bukreev · 7 months ago

Спасибо. Проблема вот в чем -- допустим, все функции next отработали без ошибок и они будут запущены асинхронно и вернут data в порядке увеличения времени (50ms, 200ms, 500ms), они в таком порядке будут попадать в стек. Пока

они не отработают, мы ничего не знаем о их data-содержании. Сложность как раз в этом -- как понять в какой последовательности они были расположены в коде функции. Вот как соблюсти порядок -- это вопрос. Буду думать...

0 Лайки

E Eugene Teaching Staff · 7 months ago

Можно это сделать через замыкание, обернув создание функции next в другую, закинув туда еще и порядковый номер

```
1  function createNext(order) {
2    return next(err, data) {}
3  }
```

0 Лайки

Alexander Bukreev · 7 months ago

спасибо, разобрался

0 Лайки

Viacheslav Evseev · 5 months ago

Сам не понял как сделал, но сделал через промисы (создаю массив промисов и сую их в Promise.all), и промисы выдают result как раз такой который и нужен для проверки последовательности.

0 Лайки

Ответить

Ответить

1

Ответить

Ответить