Олег Черний @apelsyn
Разработчик
177,1 карма 12,8 рейтинг написать
Подписаться

сегодня в 13:54

Разработка → Node.js 7.0.0 зарелизился. Встречайте async/await без babel

Node.JS*, JavaScript*

7-я нода зарелизилась, ура! Что нового:

- Движок V8 обновлён до версии 5.4.500.36, в которой обеспечена поддержка 98% возможностей JavaScript, определённых в спецификации ES2015 (ES6) и, частично, будущим стандартом ES2017.
- Отмечается новый парсер URL, соответствующий стандарту оформления URL, подготовленному сообществом WHATWG.
- Доработана работа с Buffer, Child Process, Cluster, файловой системой, промисами
- Полный список изминений

Для меня это долгожданный релиз, так как появилась возможность изпользовать конструкцию *async/await* без транспайлера babel. Включается это все ключем *--harmony*.

Теперь можно без babel так:

```
(async function() {
    let content = await require("request-promise")
                        .get("http://example.com/");
    console.log(content);
})()
```

Еще неделю назад ветка Node.js 6.x получила статус LTS, обновления для которой будут выпускаться в течение 30 месяцев. Выпуск Node.js 8 запланирован на апрель 2017 года.

Я собрал 7-ю ноду для

- CentOS 7
- Fedora 24

nodejs, node.js, javascript, async

Олег Черний @apelsyn карма 177,1 рейтинг 12,8
Разработчик

Похожие публикации

+22 NODE.JS + Windows: заглянем внутрь
15,7k 106 34

+64 Node.js для начинающих

20k 215 31

+21 Node.js v0.2.4 и ожидаемые улучшения в ветке v0.3.x

731 17 12

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+20 Идем на рекорд: пятая проверка Chromium

6,2k 13 31

+11 [В закладки] Зоопарк архитектур нейронных сетей (часть 1)

1,7k 87 7

+16 Node.js 7.0.0 зарелизился. Встречайте async/await без babel

8k 37 68

+8 Голуби брутфорсят парадокс Монти Холла лучше людей

810 7 3

+22 Сравнение библиотек логирования

3,1k 61 13

Комментарии (68)

отслеживать новые: в почте в трекере

prog666 26 октября 2016 в 13:57 +1

Я слышал что в v8 54 есть утечка памяти при использовании async await. ждем пока включат 55

ответить

rumkin 26 октября 2016 в 14:04 0

Нужны подробности. С чего вы взяли что утечку будут фиксить только в 55 версии? Возможно уже устранили или только в браузере проявляется?

ответить

Leopotam 26 октября 2016 в 14:09 *(комментарий был изменён)* 0

https://github.com/nodejs/promises/issues/4#issuecomment-254159118

ответить

kurtov 26 октября 2016 в 17:59 +1

Release V8 5.5 24 октября (два дня до выхода node 7). В этом релизе представлены async\await функции.
Баг с утечкой исправлен 16 сентября, v8 5.4 был зарелизен 9 сентября

ответить

shasoft 26 октября 2016 в 14:45 –1

А не появилась возможность запускать Node JS с флешки? Штатно, а не с помощью спец-программ.

ответить

apelsyn 26 октября 2016 в 14:54 0

Если речь идет о Windows:

- распаковываем на флешку node-v7.0.0-win-x64.zip или node-v7.0.0-win-x86.zip
- запускаем, все работает.

ответить

justboris 26 октября 2016 в 15:21 –3

> Теперь можно без babel так

Можно, но конкретно так лучше не писать.

С Promise получается короче:

```
require("request-promise")
  .get("http://example.com/")
  .then(content => console.log(content))
```

Во-вторых, с инлайновой async-функцией может возникнуть конфуз

```
(async function() {
    let content = await require("request-promise")
                        .get("http://example.com/");
    console.log("1");
```

```
})()
console.log("2");
```

В консоли будет

```
2
1
```

Что не совсем очевидно при чтении кода, хоть и понятно почему это происходит.

ответить

Yeah 26 октября 2016 в 15:27 –5

Касательно последнего примера: разве не в этом суть асинхронности? Честно говоря, меня забавляет, когда нодеры сначала долго рассказывают, как круто, что у них есть асинхроннсость, а потом придумывают костыли типа promises, чтобы таки писать синхронно.

ответить

Odrin 26 октября 2016 в 15:38 +6

Promise — это не костыль «чтобы таки писать синхронно», а способ избавиться от callback hell и структурировать асинхронный код.

ответить

HelpOP 26 октября 2016 в 16:35 +1

Вы не смотрите в корень проблемы. А конкретно почему возникает callback hell?

ответить

apelsyn 26 октября 2016 в 16:58 +1

Покажите мне цикл на промисах, чтоб я асихронно получал пользователя из базы далее что-делал полученными данными и запрашивал следующего пользователя.

Слабо?

ответить

boolive 26 октября 2016 в 17:05 –1

Промисы позволяют применить async/await или генераторы, и получите циклы с виду синхронные.

ответить

justboris 26 октября 2016 в 17:20 *(комментарий был изменён)* 0

~~А цикл бесконечный? Как запрашивается пользователь? Если у них id по порядку, то я просто могу пробежаться for-циклом и сделать запрос для каждого id.~~

~~Давайте немного не так: вы приведете пример кода с использованием async функций, а мы попробуем переписать без него без потери в читаемости.~~

UPD. Нашел ваш коммент. Попробую что-нибудь сделать.

ответить

seryh 26 октября 2016 в 17:22 0

Это делается через рекурсию.

ответить

apelsyn 26 октября 2016 в 17:27 +1

Напишите аналог на промисах через рекурсию

ответить

JustFailer 26 октября 2016 в 17:23 −1

```
const usersList = getArrayOfUsersSomehow();
let usersChain = new Promise.resolve();

usersList.foreEach( (user) => {
  usersChain = usersChain
    .then( () => doSomethingWithUser());
});
```

Что-то подобное? Не спора ради, просто интересно.

ответить

 **apelsyn** 26 октября 2016 в 17:25 +1

Напишите аналог на промисах, у вас тут "независимая" обработка кажного юзера.

ответить

 **seryh** 26 октября 2016 в 17:38 *(комментарий был изменён)* +1

```
let user,spammers = [];
(function loop() {
    getUser().then((user) => {
        isSpammer(user).then((user) => {
            spammers.push(user);
            if (spammers.length <= 10 && user !== null)
                loop();
        });
    });
}())
```

А зачем оно вам? async/await безусловно куда более лучшие инструменты чем promise.

ответить

 **Aries_ua** 26 октября 2016 в 21:21 0

> Покажите мне цикл на промисах, чтоб я асихронно получал пользователя из базы далее что-делал полученными данными и запрашивал следующего пользователя.

В данном предложении содержится просто противоречие. Вы хотите получить пользователей асинхронно, а потом синхронно что-то с ними сделать. Это как? Смысл тогда какой? Асинхронно стоит выбирать, что бы потом, так же что-то с ним сделать. Возможно я не понимаю вашу мысль. Можете более детально расписать? Я тогда сделаю для вас решение.

ответить

 **apelsyn** 26 октября 2016 в 21:27 0

В том то и дело что async/await выглядит как синхронный код а выполняется асинхронно.

Другими словами когда вы ждете ответа с базы вы не блокируете поток. В этот момент ваше приложение может обрабатвать другие запросы от пользователей.

ответить

 **enniel** 26 октября 2016 в 15:42 +3

Только вот с промисами код всё равно не становится синхронным, он остаётся асинхронным, только выглядит получше.

ответить

 **lsknwns** 26 октября 2016 в 16:41 *(комментарий был изменён)* 0

Во-первых эти «костыли» никакого отношения к нодерам не имеют, это практически **стандарт** языка.

Во-вторых, в стандарт эти «костыли» запиливаются не по желанию тех, о ком вы говорите, а как раз по желанию тех, кто не они, *чтобы облегчить им всю тягость бытия вкатывания в язык, ибо как альтернативы нет, а им так сложно разобраться и вообще во всех бедах виноват молоток, а не человек его использующий.*

Собственно если рассматривать под 'нодерами' фанатов языка, то они как раз страдают от лишней перегруженности, которая в общем-то и не нужна (cv. class), потому что 'нефанатам' *сложна*.

ответить

 **BoryaMogila** 26 октября 2016 в 15:42 +1

С промисом короче если у тебя один асинхронный запрос, а если у тебя 2 и больше асинхронных запроса, то на с async/await это:

```
(async function() {
    const request = require("request-promise")
    let someData = await request.get("http://example.com/");
    let anotherData = await request.get("http://example2.com/");
    // манипуляции с данными
    console.log(/*результат манипуляций*/);

})();
```

а с промисом:

```
const request = require("request-promise")
request
```

```
  .get("http://example.com/")
  .then((someData) => {
      request
         .get("http://example.com/")
         .then((anotherData) => {
              // манипуляции с данными
              console.log(/*результат манипуляций*/);
      })
   }
```

ничего не напоминает))

ответить

justboris 26 октября 2016 в 15:43 0

Ваш пример лучше. Мой комментарий был про то, что в статье пример использования async-функций высосан из пальца и никакой полезности не показывает.

ответить

Yeah 26 октября 2016 в 16:13 0

В вашем примере для await, если каждый запрос выполняется ровно за секунду, то через какое время выполнится console.log?

ответить

Galiaf47 26 октября 2016 в 16:23 +3

Извиняюсь если допустил ошибку, но я написал бы немного по другому

```
const request = require("request-promise")
request
  .get("http://example.com/")
  .then((someData) => {
      return request
         .get("http://example.com/")
  })
  .then((anotherData) => {
    // манипуляции с данными
```

```
    console.log(/*результат манипуляций*/);
  }
```

ответить

 **justboris** 26 октября 2016 в 16:27 +2

А я бы вообще через Promise.all сделал, чтобы было параллельно

```
const request = require("request-promise")
Promise.all([
  request.get("http://example.com/"),
  request.get("http://example2.com/")
]).then(([a ,b]) => {
  console.log(a + b);
})
```

Конечно это сработает, только если второму запросу не нужны данные первого. Если нужны, то async-functions FTW.

ответить

 **brusher** 26 октября 2016 в 17:30 +1

Если я не ошибаюсь — вы потеряли someData к моменту манипуляции с данными.
А вот вариант с Promise.all — в данном случае лучший, если только второй запрос не требует информации из первого :)

ответить

 **bertmsk** 26 октября 2016 в 16:27 +1

Это обычный promise hell. Обещали решить с применением политиканов (politicians)

ответить

 **shmidt_i** 26 октября 2016 в 16:39 0

```
const request = require("request-promise")
request
  .get("http://example.com/")
  .then((someData) => request.get("http://example.com/"))
```

```
  .then((anotherData) => {
              // манипуляции с данными
              console.log(/*результат манипуляций*/);
      })
```

В этом и смысл — сделать код более плоским

ответить

inker 26 октября 2016 в 16:39 +1

Зачем ждать, пока выполнится первый реквест, чтобы запустить второй?

```
(async function() {
  const request = require("request-promise")
  const [someData, anotherData] = await Promise.all([
    request.get("http://example.com/"),
    request.get("http://example2.com/")
  ])
  // манипуляции с данными
  console.log(/*результат манипуляций*/);
})();
```

```
const request = require("request-promise")
Promise.all([
  request.get("http://example.com/"),
  request.get("http://example2.com/")
]).then(([someData, anotherData]) => {
  // манипуляции с данными
  console.log(/*результат манипуляций*/);
})
```

ответить

walkman7 26 октября 2016 в 17:24 *(комментарий был изменён)* 0

Затем что нужны ___последовательные___ запросы (следующий запрос зависит от предыдущего).

ответить

apelsyn 26 октября 2016 в 17:29 0

Чтоб не блокировать исполнение других обработчиков. Например к вашему web-серверу пришло несколько запросов.

ответить

alek0585 26 октября 2016 в 18:33 +1

Возможно ли, что в таком случае имеет место неграмотная архитектура?

ответить

Vladimir37 26 октября 2016 в 16:39 0

Нет. С промисами вот так:

```
Promise.all([request.get("http://example.com/"), request.get("http://example2.com/")]).then((dataFirst, dataSecond) => {
        // манипуляции с данными
        console.log(/*результат манипуляций*/);
});
```

ответить

neoxack 26 октября 2016 в 17:21 0

```
(async function() {
    const request = require("request-promise")
    let someData = await request.get("http://example.com/");
    let anotherData = await request.get("http://example2.com/");
    // манипуляции с данными
    console.log(/*результат манипуляций*/);

})();
```

Лучше так:

```
let result = await Promise.all([request.get("http://example.com/"), request.get("http://example2.com/") ]);
```

ответить

 **overherz** 26 октября 2016 в 16:39 0

по-моему здесь как раз все очевидно, async/await так и должен работать

ответить

 **justboris** 26 октября 2016 в 16:59 0

Конечно очевидно, если медленно и внимательно читать 5 строчек кода.

Но представьте, как это будет выглядеть в реальном проекте, где кода больше

```
(async function() {
    let content = await require("request-promise")
                        .get("http://example.com/");
   doSomething();
   if(a !== b) {
      doSomethingElse();
   }

  // ...
  // и еще много-много кода
  // ...

  console.log("1")
})()
console.log("2");
```

Пока дочитаешь до конца, можно потерять контекст и ошибиться. А с Promise и коллбеками граница синхронного и асинхронного кода четче и понятнее.

ответить

apelsyn 26 октября 2016 в 16:48 0

А если в цикле?

ответить

justboris 26 октября 2016 в 17:00 0

А можете привести пример кода? Без него непонятно, что вы имеете в виду.

ответить

apelsyn 26 октября 2016 в 17:19 0

```
let user,spammers = [];
do {
    user = await getUser();
    if (await isSpammer(user)) {
        spammers.push(user)
    }
} while (spammers.length <= 10 && user !== null);
console.log(spammers)
```

Ну и все это, конечно, внутри асихронной функции.

ответить

justboris 26 октября 2016 в 17:33 *(комментарий был изменён)* 0

```
function collectSpammers(spammers) {
   if(spammers.length > 10) {
     return spammers;
   }
   return getUser()
     .then(user => {
       if(!user) {
         return spammers;
       }
       return isSpammer(user).then(spammer => {
```

```
            if(spammer) {
               spamers.push(users);
            }
            return spammers;
          });
        })
        .then(() => collectSpammers(spammers))
}

collectSpammers([]).then(spammers => console.log(spammers))
```

Согласен, получилось не так аккуратно как с async, но ведь реализуемо! Сомневаюсь, что нужно писать такой код регулярно, и поэтому async так уж необходим в стандарте

ответить

f0rk 26 октября 2016 в 17:58 0

Если spammers.length <= 10 && user == null он вроде бы продолжит выполняться

ответить

justboris 26 октября 2016 в 18:05 0

Почему вы так считаете? Есть же проверка сразу после getUser.

ответить

f0rk 26 октября 2016 в 18:11 0

`getUser().then(u => { if (!u) return spammers })` Зарезолвится с массивом spammers, потом вы вызываете `.then(() => collectSpammers(spammers))`, и процесс зациклился, пока getUser() возвращает null следующий в цепочке then будет вызывать collectSpammers

ответить

justboris 26 октября 2016 в 18:17 0

Действительно, спасибо за замечание! Поправил:

```
function collectSpammers(spammers) {
   if(spammers.length > 10) {
     return spammers;
   }
   return getUser().then(user => {
     if(!user) {
       return spammers;
     }
     return isSpammer(user)
       .then(spammer => {
         if(spammer) {
           spamers.push(users);
         }
         return spammers;
       })
       .then(() => collectSpammers(spammers));
   })
}

collectSpammers([]).then(spammers => console.log(spammers))
```

ответить

f0rk 26 октября 2016 в 18:52 0

Ок, раз уж пошла такая пьянка, добавлю свой вариант :)

```
function collectSpammers(n) {
  const spammers = [];

  return Array(n).fill(0).reduce(m => m
    .then(() => getUser()
      .then(function check(u) {
        if (u === null) return Promise.reject();

        return isSpammer(u).then(spammer => {
          if (spammer) spammers.push(u);
          else return getUser().then(check);
```

```
        });
      })), Promise.resolve()).then(() => spammers, () => spammers);
}

collectSpammers(10).then(spammers => console.log(spammers));
```

ответить

apelsyn 26 октября 2016 в 18:06 0

А если вам нужно проверить не случилось ли ошибки при получении данных из базы
try {
user = await getUser();
} catch (err) {
// Error handling
}
Вы представляете во что превратиться Ваш код?

async/wait это отличная конструкция для написания человек-читаемого и легко понимаемого кода. Потому что код выглядит как синхронный а виполняется асинхронно.

Ваше заявление "С Promise получается короче" работает только для примитивных конструкций, для более сложных это не всегда справедливо

ответить

justboris 26 октября 2016 в 18:12 0

> А если вам нужно проверить не случилось ли ошибки при получении данных из базы
> Вы представляете во что превратиться Ваш код?

Примерно в то же самое, во что и ваш. Добавится пара конструкций.

> Ваше заявление "С Promise получается короче" работает только для примитивных конструкций

Согласен. Но в статье вы привели именно такую конструкцию, на что я и обратил внимание. Чтобы показать хорошие стороны async можно было сразу привести пример про спаммеров, а не тривиальный request.

ответить

kurtov 26 октября 2016 в 18:39 0

```
let spammers = [];

getUser()
    .catch(onGetUserErrorHandler)
    .then(getSpammer)
    .then(() => console.log(spammers));

function getSpammer(user) {

    if (user === null) {
        return;
    }

    if (spammers.length === 10) {
        return;
    }

    return isSpammer(user)
        .then((isSpammer) => isSpammer && spammers.push(user))
        .catch(onSpammerCheckErrorHandler)
        .then(getUser)
        .catch(onGetUserErrorHandler)
        .then(getSpammer);
}
```

ответить

indestructable 26 октября 2016 в 20:18 0

then после catch не правильно, если, конечно, не хотите попасть в оба.

ответить

alek0585 26 октября 2016 в 18:42 *(комментарий был изменён)* 0

```
var user, spammers = [];

function load () {
return getUser().then(function (user) {
if (isSpammer(user)) {
spammers.push(user);
}
return user;
});
}
var array = _.map(_.range(0, 10), function () {
return load;
})
utils.sequence(array).then(function () {
console.log(spammers);
});
```

ответить

enniel 26 октября 2016 в 15:39 0

Для debian-based дистрибутивов можно установить так:

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Доку пока ещё не обновили, может и под другие дистрибутивы уже можно установить.

ответить

asdf404 26 октября 2016 в 17:52 0

Ещё можно использовать NVM:

```
nvm install v7.0.0
nvm alias default v7.0.0
```

ответить

 **boolive** 26 октября 2016 в 15:39 +1

import/export когда будет?

ответить

 **bromzh** 26 октября 2016 в 16:06 +1

А в стандарте уже прописано, как резолвить пути в этих операторах? Т.е. если мы напишем:

```
import { foo } from 'foo';
import { bar } from './bar';
```

откуда будут импортированы эти объекты?

С require всё понятно — это чисто нодовская функция, она завязана на систему модулей ноды и node_modules.

Но *стандарт* ecma не должен завязываться на какую-то конкретную реализацию и должен как-то предусмотреть различные платформы. Ведь в браузере "нет node_modules".

ответить

 **lucky_libora** 26 октября 2016 в 16:18 0

гугл на одном из недавних выступлений сказали, что пока этого в движке V8 нет

ответить

 **bromzh** 26 октября 2016 в 23:06 0

Так а разве в стандарте есть описание (хотя бы черновик) того, как именно нужно будет *резолвить* модули? Пока определились только с *синтаксисом* импортов/экспортов. А пока стандарта нет, в движок нечего добавлять.

ответить

 **lsknwns** 26 октября 2016 в 23:25 0

https://whatwg.github.io/loader/

ответить

indestructable 26 октября 2016 в 16:28 0

Резолвить по тем же правилам, что и пути в `require`. Сделали бы хоть простейшую поддержку в виде трансформации в require/module.exports.

ответить

bromzh 26 октября 2016 в 23:28 0

Можно, конечно, включить это в ноду напрямую. Люди начнут писать под ноду программы, используя нативные импорты (т.е. в npm-пакет попадёт не common-модуль (с require и module.exports), а es6-модуль с import/export).

А потом примут стандарт, и окажется, что резолв модулей как в require не совсем совместим с принятым стандартом. Что тогда делать? Уже написанные программы не будут работать на новых версиях ноды, новые программы не будут работать на тех версиях ноды, где была неправильная нативная поддержка модулей. Представьте, как весело станет управлять зависимостями, если часть из них написаны по-старой схеме, часть по-новой.

Именно для этого и есть babel. Добавить или удалить плагин в бабеле просто, а вот менять версию ноды уже тяжелее. Когда ситуация с модулями прояснится, тогда и нужно будет начинать внедрять, но не раньше.

ответить

AndreyRubankov 26 октября 2016 в 16:32 +1

Понимаю, что сейчас будут дико минусовать карму, но: Зачем?((

1. Зачем вводить функционал до того, как он появится в стандарте (это еще draft на es2017)?
2. Зачем вводить async/await на уровне языка, учитывая, что это всего лишь сахар для промисов?

вот завтра окажется, что «async/await» — это плохо, а поддерживать эту фичу придется до конца жизни ES.

ответить

staticlab 26 октября 2016 в 16:47 –1

1. Ну так реализована экспериментальная поддержка в движке v8. В самой ноде активируется только по ключу --harmony.

2. Цель — избавиться от promise hell.

ответить

AndreyRubankov 26 октября 2016 в 17:10 0

Цель благородная, но я к тому, что завтра появится async/await hell и быть беде. Если Promise API можно просто выкинуть и забыть (заменить на что-то новое), то async/await из спецификации уже не выкинешь.

Мой поинт о том, что не стоит вносить на уровень языка фичи главная цель которых сделать код красивее.

ответить

serf 26 октября 2016 в 17:15 *(комментарий был изменён)* –1

> Мой поинт о том, что не стоит вносить на уровень языка фичи главная цель которых сделать код красивее.

Хороший поинт, инженерный подход, но к сожалению вся суть развития EcmaScript относится к "деланию кода красивее", в отличии от допустим TypeScript.

ответить

dimaaan 26 октября 2016 в 23:17 0

async await впервые появился в c# в 2012 году и за 4 года не заработал себе плохой репутации.
да и вообще, как вы себе представляете async/await hell?
можно пример?

ответить

apelsyn 26 октября 2016 в 16:53 0

Это не драфт, предложение находится в Stage 3 ("Candidate"), в следующем месяце будет частью стандарта Stage 4 ("Finished").

ответить

alibertino 26 октября 2016 в 16:59 0

Потому что, по сути, это единственный путь рождения и проверки немертворожденных стандартов. А по поводу поддержки, это не обязательно, можно объявить deprecated. Поэтому и запускается не в LTS-версии.

ответить

Интересные публикации

- Голуби брутфорсят парадокс Монти Холла лучше людей 2
- Кто открыл, что Земля круглая? 1
- Компьютер «Скиапарелли», вероятно, определил успешную посадку на высоте 2–4 км 4
- Опасность и безопасность — гонка виртуальных вооружений 1
- Ученые выяснили, какие лекарства не стоит пить перед имплантацией зубов 8