7 июня 2022 г.

Привязка контекста к функции

При передаче методов объекта в качестве колбэков, например для `setTimeout`, возникает известная проблема – потеря `this`.

В этой главе мы посмотрим, как её можно решить.

Потеря «this»

Мы уже видели примеры потери `this`. Как только метод передаётся отдельно от объекта – `this` теряется.

Вот как это может произойти в случае с `setTimeout`:

```
let user = {
  firstName: "Вася",
  sayHi() {
    alert(`Привет, ${this.firstName}!`);
  }
};

setTimeout(user.sayHi, 1000); // Привет, undefined!
```

При запуске этого кода мы видим, что вызов `this.firstName` возвращает не «Вася», а `undefined`!

Это произошло потому, что `setTimeout` получил функцию `sayHi` отдельно от объекта `user` (именно здесь функция и потеряла контекст). То есть последняя строка может быть переписана как:

```
let f = user.sayHi;
setTimeout(f, 1000); // контекст user потеряли
```

Метод `setTimeout` в браузере имеет особенность: он устанавливает `this=window` для вызова функции (в Node.js `this` становится объектом таймера, но здесь это не имеет значения). Таким образом, для `this.firstName` он пытается получить `window.firstName`, которого не существует. В других подобных случаях `this` обычно просто становится `undefined`.

Задача довольно типичная – мы хотим передать метод объекта куда-то ещё (в этом конкретном случае – в планировщик), где он будет вызван. Как бы сделать так, чтобы он вызывался в правильном контексте?

Решение 1: сделать функцию-обёртку

Самый простой вариант решения – это обернуть вызов в анонимную функцию, создав замыкание:

```
let user = {
  firstName: "Вася",
  sayHi() {
    alert(`Привет, ${this.firstName}!`);
  }
};

setTimeout(function() {
  user.sayHi(); // Привет, Вася!
}, 1000);
```

Теперь код работает корректно, так как объект `user` достаётся из замыкания, а затем вызывается его метод `sayHi`.

То же самое, только короче:

```
setTimeout(() => user.sayHi(), 1000); // Привет, Вася!
```

Выглядит хорошо, но теперь в нашем коде появилась небольшая уязвимость.

Что произойдёт, если до момента срабатывания `setTimeout` (ведь задержка составляет целую секунду!) в переменную `user` будет записано другое значение? Тогда вызов неожиданно будет совсем не тот!

```
let user = {
  firstName: "Вася",
  sayHi() {
    alert(`Привет, ${this.firstName}!`);
  }
};

setTimeout(() => user.sayHi(), 1000);

// ...в течение 1 секунды
user = { sayHi() { alert("Другой пользователь в 'setTimeout'!"); } };

// Другой пользователь в 'setTimeout'!
```

Следующее решение гарантирует, что такого не случится.

Решение 2: привязать контекст с помощью bind

В современном JavaScript у функций есть встроенный метод bind, который позволяет зафиксировать `this`.

Базовый синтаксис `bind`:

```
// полный синтаксис будет представлен немного позже
let boundFunc = func.bind(context);
```

Результатом вызова `func.bind(context)` является особый «экзотический объект» (термин взят из спецификации), который вызывается как функция и прозрачно передаёт вызов в `func`, при этом устанавливая

`this=context`.

Другими словами, вызов `boundFunc` подобен вызову `func` с фиксированным `this`.

Например, здесь `funcUser` передаёт вызов в `func`, фиксируя `this=user`:

```
let user = {
  firstName: "Вася"
};

function func() {
  alert(this.firstName);
}

let funcUser = func.bind(user);
funcUser(); // Вася
```

Здесь `func.bind(user)` – это «связанный вариант» `func`, с фиксированным `this=user`.

Все аргументы передаются исходному методу `func` как есть, например:

```
let user = {
  firstName: "Вася"
};

function func(phrase) {
  alert(phrase + ', ' + this.firstName);
}

// привязка this к user
let funcUser = func.bind(user);

funcUser("Привет"); // Привет, Вася (аргумент "Привет" передан, при этом this
```

Теперь давайте попробуем с методом объекта:

```
let user = {
  firstName: "Вася",
  sayHi() {
    alert(`Привет, ${this.firstName}!`);
  }
};

let sayHi = user.sayHi.bind(user); // (*)

sayHi(); // Привет, Вася!

setTimeout(sayHi, 1000); // Привет, Вася!
```

В строке `(*)` мы берём метод `user.sayHi` и привязываем его к `user`. Теперь `sayHi` – это «связанная» функция, которая может быть вызвана отдельно или передана в `setTimeout` (контекст всегда будет правильным).

Здесь мы можем увидеть, что `bind` исправляет только `this`, а аргументы передаются как есть:

```
let user = {
  firstName: "Вася",
  say(phrase) {
    alert(`${phrase}, ${this.firstName}!`);
  }
};

let say = user.say.bind(user);

say("Привет"); // Привет, Вася (аргумент "Привет" передан в функцию "say")
say("Пока"); // Пока, Вася (аргумент "Пока" передан в функцию "say")
```

Удобный метод: `bindAll`

Если у объекта много методов и мы планируем их активно передавать, то можно привязать контекст для них всех в цикле:

```
for (let key in user) {
  if (typeof user[key] == 'function') {
    user[key] = user[key].bind(user);
  }
}
```

Некоторые JS-библиотеки предоставляют встроенные функции для удобной массовой привязки контекста, например _.bindAll(obj) в lodash.

Частичное применение

До сих пор мы говорили только о привязывании `this`. Давайте шагнём дальше.

Мы можем привязать не только `this`, но и аргументы. Это делается редко, но иногда может быть полезно.

Полный синтаксис `bind`:

```
let bound = func.bind(context, [arg1], [arg2], ...);
```

Это позволяет привязать контекст `this` и начальные аргументы функции.

Например, у нас есть функция умножения `mul(a, b)`:

```
function mul(a, b) {
  return a * b;
}
```

Давайте воспользуемся `bind`, чтобы создать функцию `double` на её основе:

```
function mul(a, b) {
  return a * b;
}

let double = mul.bind(null, 2);

alert( double(3) ); // = mul(2, 3) = 6
alert( double(4) ); // = mul(2, 4) = 8
alert( double(5) ); // = mul(2, 5) = 10
```

Вызов `mul.bind(null, 2)` создаёт новую функцию `double`, которая передаёт вызов `mul`, фиксируя `null` как контекст, и `2` – как первый аргумент. Следующие аргументы передаются как есть.

Это называется частичное применение – мы создаём новую функцию, фиксируя некоторые из существующих параметров.

Обратите внимание, что в данном случае мы на самом деле не используем `this`. Но для `bind` это обязательный параметр, так что мы должны передать туда что-нибудь вроде `null`.

В следующем коде функция `triple` умножает значение на три:

```
function mul(a, b) {
  return a * b;
}

let triple = mul.bind(null, 3);

alert( triple(3) ); // = mul(3, 3) = 9
alert( triple(4) ); // = mul(3, 4) = 12
alert( triple(5) ); // = mul(3, 5) = 15
```

Для чего мы обычно создаём частично применённую функцию?

Польза от этого в том, что возможно создать независимую функцию с понятным названием (`double`, `triple`). Мы можем использовать её и не передавать каждый раз первый аргумент, т.к. он зафиксирован с помощью `bind`.

В других случаях частичное применение полезно, когда у нас есть очень общая функция и для удобства мы хотим создать её более специализированный вариант.

Например, у нас есть функция `send(from, to, text)`. Потом внутри объекта `user` мы можем захотеть использовать её частный вариант: `sendTo(to, text)`, который отправляет текст от имени текущего пользователя.

Частичное применение без контекста

Что если мы хотим зафиксировать некоторые аргументы, но не контекст `this`? Например, для метода объекта.

Встроенный `bind` не позволяет этого. Мы не можем просто опустить контекст и перейти к аргументам.

К счастью, легко создать вспомогательную функцию `partial`, которая привязывает только аргументы.

Вот так:

```
function partial(func, ...argsBound) {
  return function(...args) { // (*)
    return func.call(this, ...argsBound, ...args);
  }
}

// использование:
let user = {
  firstName: "John",
  say(time, phrase) {
    alert(`[${time}] ${this.firstName}: ${phrase}!`);
  }
};

// добавляем частично применённый метод с фиксированным временем
user.sayNow = partial(user.say, new Date().getHours() + ':' + new Date().getMi

user.sayNow("Hello");
// Что-то вроде этого:
// [10:00] John: Hello!
```

Результатом вызова `partial(func[, arg1, arg2...])` будет обёртка `(*)`, которая вызывает `func` с:

- Тем же `this`, который она получает (для вызова `user.sayNow` – это будет `user`)
- Затем передаёт ей `...argsBound` – аргументы из вызова `partial` (`"10:00"`)
- Затем передаёт ей `...args` – аргументы, полученные обёрткой (`"Hello"`)

Благодаря оператору расширения `...` реализовать это очень легко, не правда ли?

Также есть готовый вариант _.partial из библиотеки lodash.

Итого

Метод `bind` возвращает «привязанный вариант» функции `func`, фиксируя контекст `this` и первые аргументы `arg1`, `arg2` ..., если они заданы.

Обычно `bind` применяется для фиксации `this` в методе объекта, чтобы передать его в качестве колбэка. Например, для `setTimeout`.

Когда мы привязываем аргументы, такая функция называется «частично применённой» или «частичной».

Частичное применение удобно, когда мы не хотим повторять один и тот же аргумент много раз. Например, если у нас есть функция `send(from, to)` и `from` всё время будет одинаков для нашей задачи, то мы можем создать частично применённую функцию и дальше работать с ней.

Задачи

Связанная функция как метод

важность: 5

Что выведет функция?

```
function f() {
  alert( this ); // ?
}

let user = {
  g: f.bind(null)
};

user.g();
```

решение

Ответ: `null` .

```
function f() {
  alert( this ); // null
}

let user = {
  g: f.bind(null)
};

user.g();
```

Контекст связанной функции жёстко фиксирован. Изменить однажды привязанный контекст уже нельзя.

Так что хоть мы и вызываем `user.g()` , внутри исходная функция будет вызвана с `this=null` . Однако, функции `g` совершенно без разницы, какой `this` она получила. Её единственное предназначение – это передать вызов в `f` вместе с аргументами и ранее указанным контекстом `null` , что она и делает.

Таким образом, когда мы запускаем `user.g()` , исходная функция вызывается с `this=null` .

Повторный bind

важность: 5

Можем ли мы изменить `this` дополнительным связыванием?

Что выведет этот код?

```
function f() {
  alert(this.name);
}

f = f.bind( {name: "Вася"} ).bind( {name: "Петя" } );

f();
```

решение

Ответ: **Вася**.

```
function f() {
  alert(this.name);
}

f = f.bind( {name: "Вася"} ).bind( {name: "Петя"} );

f(); // Вася
```

Экзотический объект bound function, возвращаемый при первом вызове `f.bind(...)`, запоминает контекст (и аргументы, если они были переданы) только во время создания.

Следующий вызов `bind` будет устанавливать контекст уже для этого объекта. Это ни на что не повлияет.

Можно сделать новую привязку, но нельзя изменить существующую.

Свойство функции после bind

важность: 5

В свойство функции записано значение. Изменится ли оно после применения `bind`? Обоснуйте ответ.

```
function sayHi() {
  alert( this.name );
}
sayHi.test = 5;

let bound = sayHi.bind({
  name: "Вася"
});

alert( bound.test ); // что выведет? почему?
```

решение

Ответ: `undefined`.

Результатом работы `bind` является другой объект. У него уже нет свойства `test`.

Исправьте функцию, теряющую "this"

важность: 5

Вызов `askPassword()` в приведённом ниже коде должен проверить пароль и затем вызвать `user.loginOk/loginFail` в зависимости от ответа.

Однако, его вызов приводит к ошибке. Почему?

Исправьте выделенную строку, чтобы всё работало (других строк изменять не надо).

```
function askPassword(ok, fail) {
  let password = prompt("Password?", '');
  if (password == "rockstar") ok();
  else fail();
}

let user = {
  name: 'Вася',

  loginOk() {
    alert(`${this.name} logged in`);
  },

  loginFail() {
    alert(`${this.name} failed to log in`);
  },

};

askPassword(user.loginOk, user.loginFail);
```

решение

Ошибка происходит потому, что `askPassword` получает функции `loginOk/loginFail` без контекста.

Когда они вызываются, то, естественно, `this=undefined`.

Используем `bind`, чтобы передать в `askPassword` функции `loginOk/loginFail` с уже привязанным контекстом:

```
function askPassword(ok, fail) {
  let password = prompt("Password?", '');
  if (password == "rockstar") ok();
  else fail();
}

let user = {
  name: 'Вася',

  loginOk() {
    alert(`${this.name} logged in`);
  },

  loginFail() {
```

```
    alert(`${this.name} failed to log in`);
  },

};

askPassword(user.loginOk.bind(user), user.loginFail.bind(user));
```

Теперь всё работает корректно.

Альтернативное решение – сделать функции-обёртки над `user.loginOk/loginFail`:

```
//...
askPassword(() => user.loginOk(), () => user.loginFail());
```

Обычно это также работает и хорошо выглядит. Но может не сработать в более сложных ситуациях, а именно – когда значение переменной `user` меняется между вызовом `askPassword` и выполнением `() => user.loginOk()`.

Использование частично применённой функции для логина

важность: 5

Это задание является немного усложнённым вариантом одного из предыдущих – Исправьте функцию, теряющую "this".

Объект `user` был изменён. Теперь вместо двух функций `loginOk/loginFail` у него есть только одна – `user.login(true/false)`.

Что нужно передать в вызов функции `askPassword` в коде ниже, чтобы она могла вызывать функцию `user.login(true)` как `ok` и функцию `user.login(false)` как `fail`?

```
function askPassword(ok, fail) {
  let password = prompt("Password?", '');
  if (password == "rockstar") ok();
  else fail();
}

let user = {
  name: 'John',

  login(result) {
    alert( this.name + (result ? ' logged in' : ' failed to log in') );
  }
};

askPassword(?, ?); // ?
```

Ваши изменения должны затрагивать только выделенный фрагмент кода.

решение

1.

Можно использовать стрелочную функцию-обёртку:

```
1  askPassword(() => user.login(true), () => user.login(false));
```

Теперь она получает `user` извне и нормально выполняется.

2.

Или же можно создать частично применённую функцию на основе `user.login`, которая использует объект `user` в качестве контекста и получает соответствующий первый аргумент:

```
1  askPassword(user.login.bind(user, true), user.login.bind(user, false
```

Предыдущий урок

Следующий урок

Поделиться

Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen…)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ ИЛИ ЧЕРЕЗ DISQUS ?

Имя

47 Поделиться

Лучшие Новые Старые

Юрий

месяц назад

⚡ **Хочешь понять как делать привязку контекста к функции?**
Тогда переходи в ТГ-блог «Джун на фронте»!

👨‍💻 Автор - системный администратор, который с декабря 2021 года освоил HTML, CSS, JS, Vue, Nuxt, React Native, MongoDB и Node.js.

Следи за моим путем в мир разработки: от новичка до создателя 🤖 Телеграм-бота для автоматической отправки откликов!

💡 Ежедневно ты найдешь полезные ответы на насущные вопросы, анализ рынка вакансий и советы от человека, прошедшего этот путь.

Вбивайте «Джун на фронте» и присоединяйтесь к нам!

0 0 Ответить

Baikova Maria

2 месяца назад

извините, не понимаю в этом примере
let user = {
firstName: "Вася",
sayHi() {
alert(`Привет, ${this.firstName}!`);
}
};

let sayHi = user.sayHi.bind(user); // (*)

sayHi(); // Привет, Вася!

setTimeout(sayHi, 1000); // Привет, Вася!

Понятно, что тут в любом случае будет Вася, ведь тут user.firstName не меняется. А вот если добавить в этом коде перед setTimeout: user.firstName = "Аня"; то все, в сообщении будет Аня. В чем прикол bind тут тогда?

let sayHi = user.sayHi.bind(user); // (*)

sayHi(); // Привет, Вася!

user.firstName = "Аня";

setTimeout(sayHi, 1000); // Привет, Аня!

0 0 Ответить

Дима Baikova Maria
2 месяца назад edited

потому что ты меняешь значение свойства этого объекта
То есть bind привязывает контекст (окружение), но он не запоминает объект (свойства и значения) в статическом положении, он запоминает только окружение, а все что внутри может меняться. То есть ты сначала меняешь значение свойства user.firstName, затем вызываешь функцию в контексте этого же объекта, как раз с уже измененным свойством

2 0 Ответить

Baikova Maria Дима
2 месяца назад edited

А, ну это вполне логично, да.
Я просто подумала, что тут мы также лечим ситуацию, когда произошли изменения в объекте user, но мы то в очередь поставили задание еще со старой версией.
Выходит, что с этим мы ничего сделать не можем.

Но если присвоить usery другой объект, то работает! Здорово.
Спасибо за ответ.

1 0 Ответить

Vladimir
5 месяцев назад

тема вроде должна быть непростая, как и предыдущая(ну ок, немного полегче). а задачи почему такие простые?

1 1 Ответить

Грамматический
5 месяцев назад

Чтобы решить проблему перезаписывания переменной user, можно использовать замыкания:

```
let user = {
  firstName: "Вася",
  sayHi() {
    alert(`Привет, ${this.firstName}!`);
  }
};

setTimeout((u => () => u.sayHi())(user), 1000);

user = { sayHi() { alert("Другой пользователь в 'setTimeout'!"); } };
```

Alert выведет «Привет, Вася!».

Но вариант с bind, конечно, лучше читается:

```
let user = {
  firstName: "Вася",
  sayHi() {
```

```
    alert( `Привет, ${this.firstName}!` );
  }
};
```

показать больше

0 0 Ответить

fake364
5 месяцев назад

ребят, я не понимаю. я уже больше 4 лет в разработке, и только щас узнал, что на стрелочных функциях тихо не работают bind call и apply. верней они просто ничего не делают. и почему это не пишется понятным языком предостережение? почему нигде это не пишут??

0 0 Ответить

Грамматический fake364
5 месяцев назад

В следующей главе, «Повторяем стрелочные функции», об этом сказано в разделе «Стрелочные функции VS bind».

0 0 Ответить

Грамматический fake364
5 месяцев назад

Это логично, ведь у стрелочных функций нет своего собственного this. Тем не менее, я согласен, что дополнительно подчеркнуть этот факт в этой главе было бы не лишним.

1 0 Ответить

Рулер
10 месяцев назад

решение 4 задачи

let yes = user.loginok.bind(user)
let no = user.loginReject.bind(user)

askPassword(yes,no)

0 1 Ответить

lorddop
год назад

4 задачу решила вот так:

```
askPassword.bind(user, user.loginOk, user.loginFail)();
```

1 2 Ответить

Illia K lorddop
10 месяцев назад edited

здорово, и получился тот же самый сломанный код с потерянным контекстом user, класс решение

1 0 Ответить

Молодой var
год назад edited

У меня вопрос почему в самом первом абзаце с кодом нельзя просто добавить вызов в функцию:

```
let user = {
firstName: "Вася",
sayHi() {
alert(`Привет, ${this.firstName}!`);
}
};

setTimeout(user.sayHi(), 1000); // Привет, Вася!
```

// в уроке написано setTimeout(user.sayHi, 1000);

0 0 Ответить

Sergey RJS → Молодой var
год назад

потому что ты передашь не функцию, а результат выполнения функции, так как sayHi ничего не возвращает, undefined

1 0 Ответить

Александр Пытель
год назад

В первой задаче в ответе почему-то упустили тот факт что без 'use strict' будет всё так же выводится window в браузере и global в node.js

7 0 Ответить

Egor Demeshko → Александр Пытель
год назад edited

если я правильно понял, именно для переменных, не создается новый контекс, а берется на порядок выше что ли, предыдущий. в нашем случае как раз глобал.
мы завязываемся на глобал environment record и уже от туда берем thisValue.

всеравно ничего понятно кроме того что все из
функции eval();

все. я забью, пойду дальше.

0 0 Ответить

Egor Demeshko → Александр Пытель
год назад edited

о господи. это что-то новенькое.
походу если не 'strict mode', то создается отдельный environment record для каких-то связей которые может создать функция eval().

в то время как при "strict mode" создается один record.

я думаю в современной разработке это не сильно актуально. можно забить.
хотя если невнимательно работать чисто на функциях, то можно попасть.

я всеравно, Александр, считаю, что если вы высказались, то надо давать развернутое пояснение, потому что в такой щепетильной теме можно нас неопытных легко запутать.

иногда лучше комменты не читать)))

0 0 Ответить

Egor Demeshko → Александр Пытель
год назад

вы бы обьяснили сразу почему так происходит)

0 0 Ответить

Артём
год назад

какой отстой после нормальных языков программирования

4 12 Ответить

Tazor → Артём
год назад

Ну так иди и учи нормальные языки. Что ты тут забыл?

0 2 Ответить

JS CoDER → Артём
год назад

петухонеры на месте???

1 1 Ответить

Ayur
год назад

Объясните, пожалуйста, зачем нужен второй return* в функции-обёртки, если и без него код работает?

```
function partial(func, ...argsBound) {
return function(...args) {
return func.call(this, ...argsBound, ...args); //(*)
}
}
```

0 0 Ответить

Danny Golds → Ayur
год назад

Об этом подробнее в главе "Замыкания" написано, прочитай и лучше порешай задачи на замыкания и сразу станет ясно для чего это сделано)

0 0 Ответить

Nikita Goncharov → Ayur

год назад

Чтобы новая частично применённая функция возвращала то же самое, что возвращает func. В данном случае func - метод say, который ничего не возвращает, поэтому разницы не видно.

2 0 Ответить

Sleepy Asura

год назад

Вроде как и разобрался в теме, но в тоже время не всегда понимаю к чему относится this в той или иной ситуации. Перечитывал материал уже раза 3.

7 0 Ответить

Egor Demeshko → Sleepy Asura

год назад

авторы в статьях часто пишут.

уложите пока в голове два момент использования

вызов функции в контексте обьекта через точку, практически гарантирует что this внутри функции будет ссылаться на объекта. obj.method(//тут this это obj);

и второе. если вы будете передавать метод объекта в коллбек, да хотя бы при обработке DOM события "click". можно использовать bind() или обработать событие через стрелочную функцию внутри нее указав вызов obj.method();

хорошо тому кто сразу допер как все работает)

2 0 Ответить

M

Mikhail Kiselev → Egor Demeshko

6 месяцев назад

Особенно меня восхищает это "практически"!
Жуть, для тех кто привык к норммальным языкам программирования.

0 0 Ответить

Иван

2 года назад edited

Друзья подскажите как технически (под капотом) 1-вым методом созданием функции обертки не теряется this.

```
let user = {
firstName: "Вася",
sayHi() {
alert(`Привет, ${this.firstName}!`);
}
};

setTimeout(function() {
user.sayHi(); // Привет, Вася!
}, 1000);
```

Теперь код работает корректно, так как объект user достается из замыкания, а затем вызывается его метод sayHi.

Я замыкание представляю так

показать больше

6 0 Ответить

pragmike → Иван
2 года назад edited

Я, честно говоря, тоже не понимаю, причём тут замыкание.

Мне потеря контекста видится так:

1. Если мы ссылаемся на метод объекта `user.sayHi`, то интерпретатор находит у объекта `user` свойство (в данном случае метод) `sayHi`, видит, что это функция, и возвращает этот объект функции, не глядя, какому объекту эта функция принадлежит. Это просто функция, получите, распишитесь, контекста вы не заказывали.

2. Если же мы пишем `user.sayHi()` (со скобочками), то мы говорим интерпретатору вызвать метод `sayHi` объекта `user`(исполнить его). То есть он будет знать, что нужно найти объект `user` и выполнить метод `sayHi` именно объекта user.

Попытаюсь провести аналогию. Если ты хочешь кофе, вот там кофемашина, на ней инструкция. Если ты подойдешь к машине, прочитаешь инструкцию, выполнишь её, ты получишь кофе.
Кофемашина это объект, инструкция это фукнция (метод), а ты - интерпретатор. Ты выполняешь функцию в контексте конкретной кофемашины.
Если же ты сходишь к кофемашине, возьмешь инструкцию и придешь с ней назад, ты потеряешь контекст - кофе машины уже перед тобой нет и выполнять инструкцию нет смысла. Можно применить

показать больше

8 0 Ответить

Андрей → pragmike
год назад

В том и суть замыкания, что бы найти объект user. колбэк функция ищет сначала в своем лексическом окружении, затем(если не находит) в замыкание. А замыкание это область видимости в которой функция "родилась"

0 0 Ответить

all_m1ghty_push → pragmike
год назад

```
// Судя по коду выходит следующее:
let user = {
firstName: "Вася",
sayHi() {
alert(`Привет, ${this.firstName}!`)
}
}
// Код №1:
setTimeout(user.sayHi, 1000)
```

```
// это место работает как:
// let f = user.sayHi == function() { alert(`Привет, ${this.firstName}!`) } // this == windows
// setTimeout(f, 1000)

// Код №2:
setTimeout(function() {
user.sayHi()
}, 1000)
// Или setTimeout(() => user.sayHi(), 1000)
```

показать больше

2 0 Ответить

S1m
2 года назад

Не забыть вернуться

3 0 Ответить

Михаил Виноградов
2 года назад

⚡Ну чё, народ, погнали?⚡
Реально ли **изучить javascript за 7 месяцев** и трудоустроиться?
Вот я решил проверить и веду свой блог **Джаваскриптизёр** на ютубе, где буду выкладывать видео каждую неделю на протяжении 7 месяцев.

💪💪Вторая неделя обучения завершена, второй видос тоже готов💪💪

Если тебе тоже интересен джаваскрипт, присоединяйся:
Ютуб: **Джаваскриптизёр**
ТГ: **@javascriptizerr**

🔥Удачи всем нам🔥

0 1 Ответить

Фуга
2 года назад

В последней задаче, я как бы догадался, какой верный ответ и написал. Но так и не понял, как result получает значение.

0 0 Ответить

Фуга → Фуга
2 года назад

А, понял. При вызове функции, аргумент в bind присваивается функции, которую мы и вызываем.

3 0 Ответить

sudo mkfs -t vfat /dev/sdb1
2 года назад edited

Подарочек, обертка-заменитель prompt на command line для последней задачки кто на ноде поиграться

хочет 🤖

```
import { createInterface } from 'readline';
const rl = createInterface({
    input: process.stdin,
    output: process.stdout
});

function askPassword(ok, fail) {
    rl.question('Password? ', function (pass) {
        let password = pass;
        if (password == "123") ok();
        else fail();
        rl.close();
    });
    rl.on('close', function () {
        process.exit(0);
    });
}

let user = {
    name: 'John',

    login(result) {
        console.log(this.name + (result ? ' logged in' : ' failed to log in'));
    }
};

askPassword(?, ?); // ?
```

0 0 Ответить

Tony Dias
2 года назад

Если проходить в день по 1 теме, то вчера был экзамен, сегодня - выходной (=

1 0 Ответить

Иван Tony Dias
2 года назад

да, прошлая тема жесткая была.

1 0 Ответить

Дима Дим Tony Dias
2 года назад

а сегодня что у тебя?

0 0 Ответить

Konstantin Aleksandrov
2 года назад

Uncaught TypeError: can't access property "bind", user.login() is undefined

выдало эту ошибку, когда решил последнюю задачу. Окно prompt не отработало...

1 0 Ответить

Danny Golds Konstantin Aleksandrov
год назад

может быть, ты каким-то чудом объект не тот создал или метод...

0 0 Ответить

Eugene Klevtsov
2 года назад edited

Чёт мне подсказывает, что для фронтендера это неособо нужная штука... :)

2 1 Ответить

Danny Golds → Eugene Klevtsov
год назад

особо нужная, просто когда небольшой проект, то да, а когда у тебя сайт-магазин, то такая штука необходима и такое повсеместно применяется в различных библиотеках.

0 0 Ответить

Владимир Филиппов
2 года назад

> Таким образом, для this.firstName он пытается получить window.firstName, которого не существует.

Создал переменную firstName (let firstName) вне объекта - undefined
создал переменную без let (firstName = "Gena") вне объекта - переменная нашлась.

Это как так ? let изменяет window ?

0 0 Ответить

Danny Golds → Владимир Филиппов
год назад

без ключевого слова let/const/var невозможно создать переменную в глобальной части скрипта, будет ошибка, а вот var записывает переменную в глобальный объект window/global

0 0 Ответить

Ilya → Владимир Филиппов
2 года назад

"особенность" JavaScript в свободном режиме, присвоение совершенно необъявленного идентификатора не является ошибкой; вместо этого он создает свойства глобального объекта, а свойства глобального объекта являются глобальными переменными. (До ES5 все глобальные переменные были свойствами глобального объекта. Начиная с ES2015, был добавлен новый тип глобальных объектов, который не является свойством глобального объекта.

1 0 Ответить

Ilya → Владимир Филиппов
2 года назад

В браузере глобальные функции и переменные, объявленные с помощью var (не let/const!), становятся свойствами глобального объекта. Глава 6.5 (Глобальный объект)

0 0 Ответить

Andrew Sokolovsky → Владимир Филиппов
2 года назад

По идее не должно находиться, создавать переменную без let вообще не правильно(ведь сначала она создается с let/const/(var) , а потом ты ее например изменяешь с "=") можешь скинуть код

0 0 Ответить

vitrums
2 года назад

Мой Node.js и Chrome в первой задаче логируют globalThis, а не null. Просьба к администраторам сайта пояснить, с чем связано такое повдение скрипта на текущий момент.

0 0 Ответить

Олексій Ловченко → vitrums
2 года назад

Я никакое отношение к администрации не имею, но у меня в VSCode пишет null. Попробуйте запустить с use strict

0 0 Ответить

vitrums → Олексій Ловченко
2 года назад

Спасибо! С 'use strict' действительно в дальнейшем this находится в значении, которое передано как первый аргумент bind.

0 0 Ответить

WASD
2 года назад

AAAAA 26.07.2022
Я сделал последнюю задачу сам через
askPassword(user.login.bind(user, true), user.login.bind(user, false))

8 0 Ответить

Дима Дим → WASD
2 года назад

них, крутяк. Я когда прочитал решение то тож подумал что не так и сложно то

1 0 Ответить

WASD → Дима Дим
2 года назад

Прикольно, был тут 2 месяца назад и хочу тебе сказать что тебя ждет многое)
Не бросай и все будет)) удачи тебе
PS: Сейчас перечитываю учебник и много что вспоминаю.

0 0 Ответить

Lessletter
2 года назад

Тем, кто как и я не понимал зачем столько функций привязки (call, apply, bind, bindAll):

call, apply - ВЫЗЫВАЮТ на месте эту же функцию с привязкой.
bind, bindAll - создают копию функции с привязкой, для вызова в нужное время.

знаю что это очевидно для всех, но не для меня) так что без хейта)

30 0 Ответить

namediablo Lessletter
2 месяца назад edited

Более подробно, если кто не понял
Метод bind применяется для создания копии функции с привязкой к определенному объекту. Это позволяет установить контекст выполнения функции, то есть значение this внутри функции будет ссылаться на указанный объект. Bind не вызывает функцию сразу, а возвращает новую функцию, которую вы можете вызвать позже.

Методы call и apply, напротив, вызывают функцию сразу. Они позволяют вам передавать аргументы в функцию и указывать объект, на который будет ссылаться this внутри этой функции. Главное различие между call и apply заключается в том, что в методе call аргументы передаются как список, а в методе apply - как массив.

Применение bind, call и apply зависит от ситуации. Bind используется, когда нужно сохранить определенный контекст выполнения функции. Call и apply удобны, когда требуется вызвать функцию сразу с определенным контекстом и аргументами.

Короче, отличие в том, что bind создает копию. Это безопасно есть ли в основном обьекте функцию могут изменить. А call и apply экономят ресурсы.

показать больше

0 0 Ответить

aleksversus Lessletter
год назад

Можешь увидеть по лайкам, что это вообще не очевидно. Спасибо огромное, дружище.

1 0 Ответить

Anna Lessletter
2 года назад

Мне было не очевидно, так что спасибо :)

0 0 Ответить

Кирилл
2 года назад

Мне не понятны пункты (1) и (2): запись ...args в параметре функции в (1) - это массив, зачем в (2) вызывается func с остаточным параметром ...args, если args есть уже массив?

```
function partial(func, ...argsBound) {
  return function(...args) { // (1)
    return func.call(this, ...argsBound, ...args); // (2)
```

```
  }
}

// использование:
let user = {
  firstName: "John",
  say(time, phrase) {
    alert(`[${time}] ${this.firstName}: ${phrase}!`);
  }
};

user.sayNow = partial(user.say, new Date().getHours() + ':' + new Date().getMinutes());

user.sayNow("Hello", "Hi");
```

1 0 Ответить

Maxim → Кирилл

2 года назад edited

...args и **...argsBound** в параметрах - это синтаксис **rest**, используется он исключительно в параметрах функций. По сути вы говорите, что вызов функции начиная с параметра **...args**, будет накапливать в себе аргументы. Следовательно вы не можете писать что-то такое **(function(...rest1, ...rest2){})()** - так как такой синтаксис запрещен (так как неизвестно сколько должно уходить аргументов в первый параметр и сколько во второй, делить пополам - не рационально).

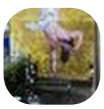
...argsBound и **..args** в аргументах - это синтаксис **spread**. Он распаковывает массив в список аргументов для функции. Множественность **spread** разрешена вы можете иметь условно несколько массивов которые хотите чтобы их содержимое стало аргументами для функции.

Что касается конструкции func.call(**this, ...argsBound, ...args**). Функция Function.prototype.call принимает первым аргументом контекст функции, а последующими (необязательно) аргументы для вызываемой функции. Отсюда мы можем понять что **...argsBound** это набор значений из замыкания то есть из функции **partial**, а **...args** это набор значений из функции которую мы вызываем прямо сейчас. В итоге у нас есть аргументы которые закешированы замыканием и аргументы, которые мы передаем в данный момент времени.

Ясно?!

показать больше

6 0 Ответить



Хоц → Maxim

2 года назад

Не могу разобраться в чем разница использования замкнутой функции в обертке в сравнении использования функции с привязкой bind?
Какая разница применения методов user.sayNow и user.sayNow2 в этом случае?

```
  function partial(func, ...argsBound) {
   return function(...args) { // (*)
     return func.call(this, ...argsBound, ...args);
   }
 }
```

```
// использование:
let user = {
  firstName: "John",
  say(time, phrase) {
    alert(`[${time}] ${this.firstName}: ${phrase}!`);
  }
};

// добавляем частично применённый метод с фиксированным временем
user.sayNow = partial(user.say, new Date().getHours() + ':' + new Date().getMinutes());
user.sayNow("Hello");

user.sayNow2 = user.say.bind(user, new Date().getHours() + ':' + new Date().getMinutes());
user.sayNow2('hello');
```

0 0 Ответить

Maxim → Хоц

2 года назад edited

Если вы читали мое сообщение насчет работы **bind**, то могли заметить пример с замыканием:

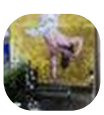
```
function bindX(func, thisArg, ...boundArgs){
  if(typeof func !== "function") throw Error("func is not a callable");
  const closure = function(...args){
     if(new.target){
       return Reflect.construct(func, [...boundArgs, ...args], new.target);
     }
     return Reflect.apply(func, thisArg, [...boundArgs, ...args])
  }
  Object.setPrototypeOf(closure, Object.getPrototypeOf(func));
  return closure;
}
```

Это практический эквивалент метода **Function.prototype.bind**. Так что ответ на ваш вопрос очень простой: разница в реализациях. **Function.prototype.bind** это внутренняя реализация, а ваш **partial** или мой **bindX** это собственная реализация.

Но я бы не советовал так писать собственную реализацию, так как в вашем **partial** в возвращаемой функции вы опираетесь на **this**, что не есть хорошо, так как func.call будет опираться на контекст вызова возвращаемой функции. В ваших примерах все хорошо так как вы эту возвращаемую функцию присваиваете в свойство объекта, а потом вызываете как метод объекта. Но попробуйте вызвать ее как функцию то контекст будет потерян, что противоречит привязки контекста.

показать больше

1 0 Ответить



Хоц → Maxim

2 года назад

Спасибо, за столь обширный ответ, помогли!

0 0 Ответить

К **Кирилл** → Maxim

2 года назад

Понял, спасибо

1 0 Ответить

Maxim

2 года назад edited

Предупреждение: ***внимательно смотрите за ссылками, несмотря на то что они имеют одинаковые имена, они могут указывать на разные алгоритмы***

Смотрю я как-то на эту статью и даже как-то печально, что отсутствует объяснение на счет экзотического вызова функции **bind**.

Давайте я расскажу чем отличается обычный вызов функции от экзотического вызова функции **bind**.

Смотрите, у обычной функции, которую вы обычно определяете в своем коде, при ее создании создаются для нее специфические поля **[[Call]]** (для любой функции) и **[[Construct]]** (только для обычных функций, функций-классов, функций созданных через конструктор **Function**). После создания, обычный вызов функции пользуется полем **[[Call]]**, при создании экземпляра используется **[[Construct]]**.

- Вызов функции: `a();`
- Создание экземпляра: `new a();`

История с функцией, которая создает функцию **bind** аналогичная, но есть одно но. Эта функция при создании определяет другое поведение для **[[Call]]** и **[[Construct]]**. Но помимо этих полей есть **список дополнительных полей**, которые содержат: функцию, которую мы на самом деле хотим вызвать,

показать больше

16 0 Ответить

Александр Вельможко → Maxim

2 года назад

Жаль только я не понимаю, что ты пишешь(

6 0 Ответить

bi_zi → Maxim

2 года назад

Maxim как всегда на высоте) спасибо за твой вклад в это сообщество

2 0 Ответить

Maxim → bi_zi

2 года назад edited

Спасибо за приятные слова :)

2 0 Ответить

Arystan

2 года назад

1. null
2. Вася
3. undefined // создается bound - другой объект
4. askPassword(user.loginOk.bind(user), user.loginFail.bind(user))
5. askPassword(user.login.bind(user, true), user.login.bind(user, false))

0 0 Ответить

W **wissem**
2 года назад

Но ведь в первой задаче не null будет, а window. И только в строгом редиме null. Неплохо было бы об этом напоминать.

0 0 Ответить

Y **Yura Kulikov**
2 года назад

Господа инопланетяне поясните , пожалуйста эти строчки(пример с функцией partial):
Тем же this, который она получает (для вызова user.sayNow – это будет user)
Затем передаёт ей ...argsBound – аргументы из вызова partial ("10:00")
Затем передаёт ей ...args – аргументы, полученные обёрткой ("Hello"). Видимо я настолько туп ,что не понимаю, в особенности: ...args – аргументы, полученные обёрткой ("Hello").Откуда ,куда , зачем....

0 0 Ответить

G **GAEM** → Yura Kulikov
2 года назад

say(time, phrase) функция получает два аргумента, ...argsBound как раз таки записывается в time.
partial возвращает фукнцию, которая принимает один аргумент, который как раз таки и будет ...args (хотя не понятно почему именно ...args, а не какая нибудь обычная переменная).
Ну и затем эта возвращаемая функция, вовращает функцию переданную в первом аргументе partial (func, в данном случае это say(time, phrase)). Возвращает уже с принятыми аргументами say(...argsBound, ...args)

1 0 Ответить

B **Вячеслав Серпиченко**
2 года назад

Скопировал код из 4-й задачи и у меня нет никакой ошибки. Может кто подсказать с чем это связано? Обнова языка? Да и в первой задаче у меня выводится не null а объект window

0 0 Ответить

H **Никита Смердов** → Вячеслав Серпиченко
2 года назад

Потому что у тебя не строгий режим стоит. Поменяй на строгий и работать перестанет

0 0 Ответить

P **Руслан Моцуков** → Вячеслав Серпиченко
2 года назад edited

точно выводит 'Вася logged in' ? или выдает, без имени 'logged in'?

0 0 Ответить

M **Михаил Крутов**
2 года назад

Приветствую, может кто-нибудь подсказать, как подключать библиотеки к проекту. Я пытаюсь подключить lodash, делаю как в инструкции, и webstorm видит библиотеку, подсказки методов отображаются, но когда пытаюсь вывести на сайт отладчик браузера сообщает, что не знает методов библиотеки.
Если добавляю библиотеку отдельным файлом в папку проекта, то в консоли браузера появляются эти методы, но при этом, когда я использую методы в файле .js привязаном к проекту, браузер пишет, что не вкурсе, что это за метод

0 0 Ответить

E **Egorka** → Михаил Крутов
2 года назад

вероятно вы сначала вызываете методы библиотеки, а затем она по коду подключается, а не наоборот

0 0 Ответить

A **Александр Сергеев**
2 года назад

Благодаря оператору расширения ... реализовать это очень легко, не правда ли?
Ответ: Нет, не правда.

11 4 Ответить

Y **Yura Kulikov** → Александр Сергеев
2 года назад

Согласен, эта часть сложна в понимании.Тяжко будет дальше совсем чую...

1 0 Ответить

A **Andrew Sokolovsky** → Yura Kulikov
2 года назад

как вариант используйте разные источники. На ютубе про bind и spread operator можно найти

0 0 Ответить

Загрузить ещё комментарии