6 января 2023 г.

Копирование объектов и ссылки

Одно из фундаментальных отличий объектов от примитивов заключается в том, что объекты хранятся и копируются «по ссылке», тогда как примитивные значения: строки, числа, логические значения и т.д. – всегда копируются «как целое значение».

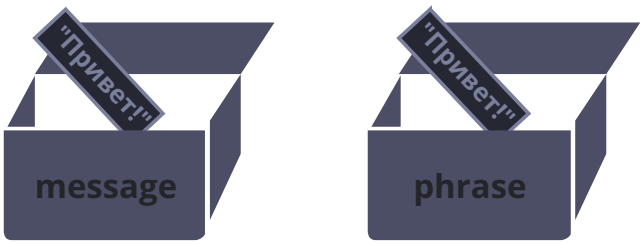
Это легко понять, если мы немного заглянем под капот того, что происходит, когда мы копируем значение.

Давайте начнём с примитива, такого как строка.

Здесь мы помещаем копию `message` во `phrase`:

```
1  let message = "Привет!";
2  let phrase = message;
```

В результате мы имеем две независимые переменные, каждая из которых хранит строку `"Привет!"`.



Вполне очевидный результат, не так ли?

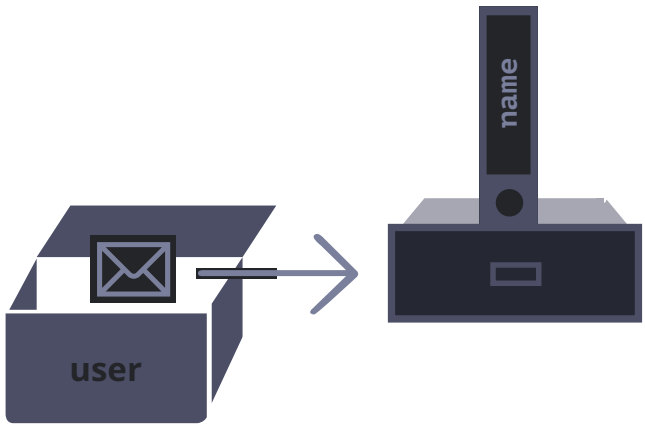
Объекты ведут себя иначе.

Переменная, которой присвоен объект, хранит не сам объект, а его «адрес в памяти» – другими словами, «ссылку» на него.

Давайте рассмотрим пример такой переменной:

```
1  let user = {
2    name: "John"
3  };
```

И вот как это на самом деле хранится в памяти:



Объект хранится где-то в памяти (справа от изображения), в то время как переменная `user` (слева) имеет лишь «ссылку» на него.

Мы можем думать о переменной объекта, такой как `user`, как о листе бумаги с адресом объекта на нем.

Когда мы выполняем действия с объектом, к примеру, берём свойство `user.name`, движок JavaScript просматривает то, что находится по этому адресу, и выполняет операцию с самим объектом.

Теперь вот почему это важно.

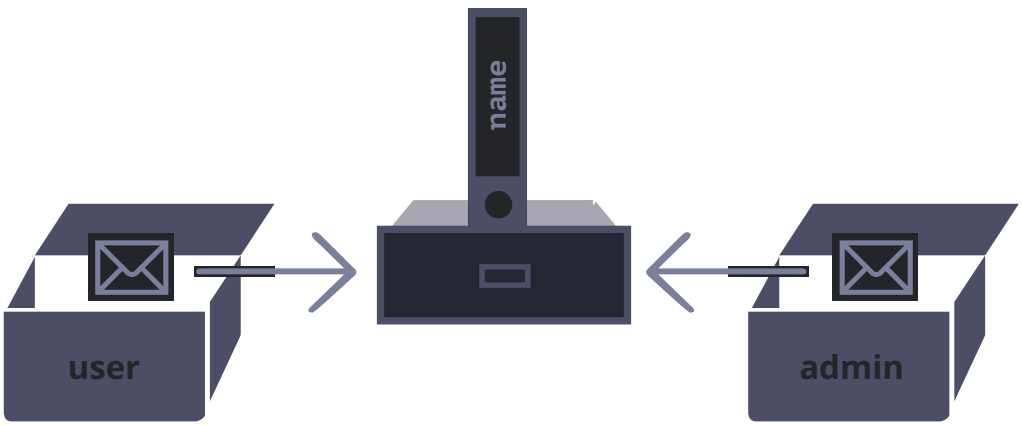
При копировании переменной объекта копируется ссылка, но сам объект не дублируется.

Например:

```
let user = { name: "John" };

let admin = user; // копируется ссылка
```

Теперь у нас есть две переменные, каждая из которых содержит ссылку на один и тот же объект:



Как вы можете видеть, все ещё есть один объект, но теперь с двумя переменными, которые ссылаются на него.

Мы можем использовать любую переменную для доступа к объекту и изменения его содержимого:

```
let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // изменено по ссылке из переменной "admin"
```

```
6
7 alert(user.name); // 'Pete', изменения видны по ссылке из переменной "user"
```

Это как если бы у нас был шкафчик с двумя ключами, и мы использовали один из них (`admin`), чтобы войти в него и внести изменения. А затем, если мы позже используем другой ключ (`user`), мы все равно открываем тот же шкафчик и можем получить доступ к изменённому содержимому.

Сравнение по ссылке

Два объекта равны только в том случае, если это один и тот же объект.

Например, здесь `a` и `b` ссылаются на один и тот же объект, поэтому они равны:

```
1 let a = {};
2 let b = a; // копирование по ссылке
3
4 alert( a == b ); // true, обе переменные ссылаются на один и тот же объект
5 alert( a === b ); // true
```

И здесь два независимых объекта не равны, даже если они выглядят одинаково (оба пусты):

```
1 let a = {};
2 let b = {}; // два независимых объекта
3
4 alert( a == b ); // false
```

Для сравнений типа `obj1 > obj2` или для сравнения с примитивом `obj == 5` объекты преобразуются в примитивы. Очень скоро мы изучим, как работают преобразования объектов, но, по правде говоря, такие сравнения требуются очень редко и обычно они появляются в результате ошибок программиста.

Клонирование и объединение, Object.assign

Итак, копирование объектной переменной создаёт ещё одну ссылку на тот же объект.

Но что, если нам всё же нужно дублировать объект? Создать независимую копию, клон?

Это тоже выполнимо, но немного сложнее, потому что в JavaScript для этого нет встроенного метода. Но на самом деле в этом редко возникает необходимость, копирования по ссылке в большинстве случаев вполне хватает.

Но если мы действительно этого хотим, то нам нужно создать новый объект и воспроизвести структуру существующего, перебрав его свойства и скопировав их на примитивном уровне.

Например так:

```
1 let user = {
2   name: "John",
3   age: 30
4 };
5
6
7
8
9
```

```
let clone = {}; // новый пустой объект

// давайте скопируем все свойства user в него
for (let key in user) {
  clone[key] = user[key];
}

// теперь clone это полностью независимый объект с тем же содержимым
clone.name = "Pete"; // изменим в нём данные

alert( user.name ); // все ещё John в первоначальном объекте
```

Также мы можем использовать для этого метод Object.assign.

Синтаксис:

```
Object.assign(dest, [src1, src2, src3...])
```

- Первый аргумент `dest` — целевой объект.
- Остальные аргументы `src1, ..., srcN` (может быть столько, сколько необходимо) являются исходными объектами
- Метод копирует свойства всех исходных объектов `src1, ..., srcN` в целевой объект `dest`. Другими словами, свойства всех аргументов, начиная со второго, копируются в первый объект.
- Возвращает объект `dest`.

Например, мы можем использовать его для объединения нескольких объектов в один:

```
let user = { name: "John" };

let permissions1 = { canView: true };
let permissions2 = { canEdit: true };

// копируем все свойства из permissions1 и permissions2 в user
Object.assign(user, permissions1, permissions2);

// теперь user = { name: "John", canView: true, canEdit: true }
```

Если скопированное имя свойства уже существует, оно будет перезаписано:

```
let user = { name: "John" };

Object.assign(user, { name: "Pete" });

alert(user.name); // теперь user = { name: "Pete" }
```

Мы также можем использовать `Object.assign` для замены цикла `for..in` для простого клонирования:

```
let user = {
  name: "John",
  age: 30
```

```
};

let clone = Object.assign({}, user);
```

Он копирует все свойства `user` в пустой объект и возвращает его.

Также существуют и другие методы клонирования объекта. Например, с использованием оператора расширения `clone = {...user}`, рассмотренного далее в учебнике.

Вложенное клонирование

До сих пор мы предполагали, что все свойства `user` примитивные. Но свойства могут быть и ссылками на другие объекты. Что с ними делать?

Например, есть объект:

```
let user = {
  name: "John",
  sizes: {
    height: 182,
    width: 50
  }
};

alert( user.sizes.height ); // 182
```

Теперь недостаточно просто скопировать `clone.sizes = user.sizes`, потому что `user.sizes` – это объект, он будет скопирован по ссылке. Таким образом, `clone` и `user` будут иметь общий объект `sizes`:

```
let user = {
  name: "John",
  sizes: {
    height: 182,
    width: 50
  }
};

let clone = Object.assign({}, user);

alert( user.sizes === clone.sizes ); // true, тот же объект

// user и clone обладают общим свойством sizes
user.sizes.width++;       // изменяем свойства в первом объекте
alert(clone.sizes.width); // 51, видим результат в другом
```

Чтобы исправить это, мы должны использовать цикл клонирования, который проверяет каждое значение `user[key]` и, если это объект, тогда также копирует его структуру. Это называется «глубоким клонированием».

Мы можем реализовать глубокое клонирование, используя рекурсию. Или, чтобы не изобретать велосипед заново, возьмите готовую реализацию, например _.cloneDeep(obj) из библиотеки JavaScript lodash.

Также мы можем использовать глобальный метод structuredClone(), который позволяет сделать полную копию объекта. К сожалению он поддерживается только современными браузерами. Здесь можно ознакомиться с поддержкой этого метода.

Объекты, объявленные как константа, могут быть изменены

Важным побочным эффектом хранения объектов в качестве ссылок является то, что объект, объявленный как `const`, *может* быть изменён.

Например:

```
const user = {
  name: "John"
};

user.name = "Pete"; // (*)

alert(user.name); // Pete
```

Может показаться, что строка `(*)` вызовет ошибку, но, это не так. Значение `user` это константа, оно всегда должно ссылаться на один и тот же объект, но свойства этого объекта могут свободно изменяться.

Другими словами, `const user` выдаст ошибку только в том случае, если мы попытаемся задать `user=...` в целом.

Тем не менее, если нам действительно нужно создать постоянные свойства объекта, это тоже возможно, но с использованием совершенно других методов. Мы затронем это в главе Флаги и дескрипторы свойств.

Итого

Объекты присваиваются и копируются по ссылке. Другими словами, переменная хранит не «значение объекта», а «ссылку» (адрес в памяти) на это значение. Таким образом, копирование такой переменной или передача её в качестве аргумента функции копирует эту ссылку, а не сам объект.

Все операции с использованием скопированных ссылок (например, добавление/удаление свойств) выполняются с одним и тем же объектом.

Чтобы создать «реальную копию» (клон), мы можем использовать `Object.assign` для так называемой «поверхностной копии» (вложенные объекты копируются по ссылке) или функцию «глубокого клонирования», такую как _.cloneDeep(obj).

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.

Сергей Кирилюк
3 дня назад

Кто вы, кто разбирается в этом?! Тот кто придумал, походу бог.

0 0 Ответить • Поделиться ›

Dimon Nikolaew
24 дня назад edited

```
const a = {
j: {
m: 'love'
}
}

const b = JSON.parse(JSON.stringify(a));

a.j.m = 'death';

console.log(b.j.m); // love
```

0 0 Ответить • Поделиться ›

HymeraDoma
месяц назад

deepClone:

```
let user = {
name: `John`,
age: 32,
g:{
f: 3,
d:4,
}
};
function deepClone(obj) {
let clObj = {};
for(let i in obj) {
if (typeof obj[i] === `object`) {
clObj[i] = deepClone(obj[i]);
```

```
}
let user2 = deepClone(user);

alert(user2===user);
```

правильно?

0 0 Ответить • Поделиться ›

Artem Dots

2 месяца назад edited

```
function cloneObj(obj){

//Создается пустой объект "клон"
let clone = {}

//Перебор каждого свойства "клонируемого"
for (let prop in obj) {

// Если тип значения свойства - объект, то значению свойства "клона" присваивается результат рекурсивно вызванной этой же функции, но с аргументом значения свойства "клонируемого"
if (typeof obj[prop] == 'object') {
clone[prop] = cloneObj(obj[prop])
}

// В противном случае значению свойства "клона" присваивается соответствующее значение "клонируемого"
else {
clone[prop] = obj[prop]
}
```

показать больше

5 0 Ответить • Поделиться ›

Hick Hickert → Artem Dots

месяц назад

Круто...
только проверка глубже уже не дает такого результата:

```
//...
menu2.sizes.relative.padding.top = 777;
console.log('menu: ', menu2.sizes.relative.padding.top); // menu:  777
console.log('menu2: ', menu2.sizes.relative.padding.top); //menu2:  777
```

0 1 Ответить • Поделиться ›

Vladimir Degtyarev

2 месяца назад

> Вполне очевидный результат, не так ли?

С чего бы он очевиден? Переменные это коробки со значениями, а коллекции это коробки с сылками. Вот в python и там и там ссылки. Это очевидно, логично и понятно) Тут нет

0 3 Ответить • Поделиться ›

хлорка
3 месяца назад

13.12.2022 19:34

0 0 Ответить • Поделиться ›

Алексей Олейник → хлорка
2 месяца назад

Широта какая?)

3 0 Ответить • Поделиться ›

Ихтияр Кадыров
4 месяца назад

Код для глубокого клонирования.

```
const OBJECT_T = "[object Object]";
   const MAP_T = "[object Map]";
   const SET_T = "[object Set]";
   const DATE_T = "[object Date]";

   function getTypeOf(context) {
       let _toString = Object.prototype.toString;
       return _toString.call(context);
   }

   function cloneDeep(obj, clone = {}) {

       for (let key in obj) {

           let currentType = getTypeOf(obj[key]);

           if (Array.isArray(obj[key])) {
               clone[key] = obj[key].slice();
           } else if ( currentType === OBJECT_T) {
               clone[key] = {};
               cloneDeep(obj[key], clone[key]);
           } else if (currentType === MAP_T) {
               clone[key] = new Map(obj[key]);
           } else if (currentType === SET_T) {
               clone[key] = new Set(obj[key]);
           } else if (currentType === DATE_T) {
               clone[key] = new Date(obj[key]);
           } else {
               clone[key] = obj[key];
           }
       }

       return clone;
   }
```

3 3 Ответить • Поделиться ›

Sleepy Asura → Ихтияр Кадыров
3 месяца назад

Bruh.

5 0 Ответить • Поделиться ›

Наталья
4 месяца назад edited

Глубокое клонирование через рекурсию

Глубокое клонирование через рекурсию

```
const man = {
name: 'John',
activity: { specialist: { engineer: 'tester' } },
slogan: () => { console.log('Is only forward!') }
};
const man1 = {};

function clonObj(newObj, oldObj) {
for (let prop in oldObj) {
if (typeof oldObj[prop] === 'object') {
newObj[prop] = {};
clonObj(newObj[prop], oldObj[prop])
}
else {
newObj[prop] = oldObj[prop]
};
};
};
clonObj(man1, man);
console.log(man1);
```

4 0 Ответить • Поделиться ›

ThisIsSparta
5 месяцев назад

const dest = structuredClone(source);
В современном JS делает глубокую копию, статья устаела

12 0 Ответить • Поделиться ›

lozzka ThisIsSparta
3 месяца назад

Спасибо!

0 0 Ответить • Поделиться ›

Скалозуб Василий ThisIsSparta
4 месяца назад

он не копирует функции

1 0 Ответить • Поделиться ›

Roman Chernyshcov ThisIsSparta
4 месяца назад

Да, все супер работает. Пасиб!

0 0 Ответить • Поделиться ›

Fogg Gustavson ThisIsSparta
5 месяцев назад

Когда функция была добавлена?

1 0 Ответить • Поделиться ›

Дарья Янукович
6 месяцев назад

В компании мы создали площадку, где решаем и изучаем js

Переходи к нам в ТГ deveveloper_house_jun_front.

0 2 Ответить • Поделиться ›

Oleksii Pishchuhin
6 месяцев назад

говно статья
как весь обьект скопировать понятно

как достать и использовать конкретное отдельное свойство , не по ссылке
как скопировать одно свойство - не понятно

1 1 Ответить • Поделиться ›

GROL COON → Oleksii Pishchuhin
5 месяцев назад edited

Если свойство - примитив, то можешь "достать" его простым обращением к нему.
good = {cost: 5, value: 'skirt'};
let item = good.value; //item теперь равен 'skirt'
item= 'blue '+item; // item теперь равен 'blue skirt'
let price = good.cost; //price теперь равен 5
price = price * 2; //price теперь равен 10
console.log(`Ваш предмет: ${item}, Ваша цена: ${price}`); //Ваш предмет blue skirt, Ваша цена 10

При этом значение объекта good осталось неизменным.

P.S. надеюсь правильно понял вопрос и дал понятный ответ.

2 1 Ответить • Поделиться ›

Karpenko Evgenuj → GROL COON
5 месяцев назад

const user ={name: "Vasja", age: 35, company: "dev"}
есть объект юзер, нужно сделать новый объект на основе юзера, но скопировать только имя и компанию.

1 0 Ответить • Поделиться ›

Загрузить ещё комментарии