Раздел
Объекты: основы
Навигация по уроку
Сравнение по ссылке
Клонирование и объединение объектов, Object.assign
Вложенное клонирование
Итого
Комментарии
Поделиться
Редактировать на GitHub

Копирование объектов и ссылки

Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке».

Примитивные типы: строки, числа, логические значения – присваиваются и копируются «по значению».

Например:

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

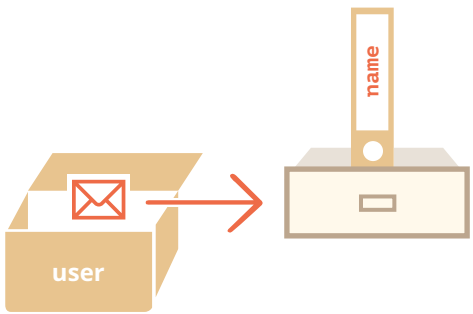
В результате мы имеем две независимые переменные, каждая из которых хранит строку `"Привет!"`.

Объекты ведут себя иначе.

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

Проиллюстрируем это:

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



Сам объект хранится где-то в памяти. А в переменной `user` лежит «ссылка» на эту область памяти.

Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.

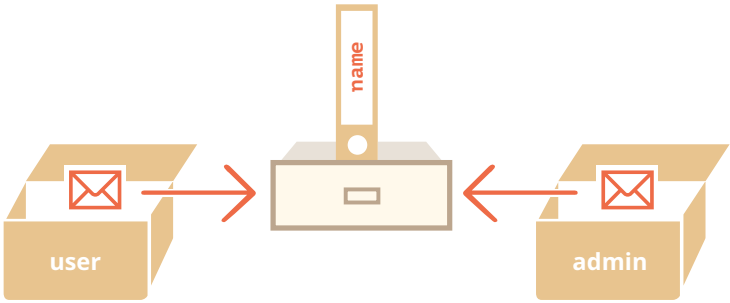
Если мы представляем объект как ящик, то переменная – это ключ к нему. Копирование переменной дублирует ключ, но не сам ящик.

Например:

```
let user = { name: "Иван" };

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

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



Мы можем использовать любую из переменных для доступа к ящику и изменения его содержимого:

```
let user = { name: 'Иван' };

let admin = user;

admin.name = 'Петя'; // изменено по ссылке из переменно

alert(user.name); // 'Петя', изменения видны по ссылке
```

Приведённый выше пример демонстрирует, что объект только один. Как если бы у нас был один ящик с двумя ключами и мы использовали один из них (`admin`), чтобы войти в него и что-то изменить, а затем, открыв ящик другим ключом (`user`), мы бы увидели эти изменения.

Сравнение по ссылке

Операторы равенства `==` и строгого равенства `===` для объектов работают одинаково.

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

В примере ниже две переменные ссылаются на один и тот же объект, поэтому они равны друг другу:

```
let a = {};
let b = a; // копирование по ссылке

alert( a == b ); // true, т.к. обе переменные ссылаются
alert( a === b ); // true
```

В другом примере два разных объекта не равны, хотя оба пусты:

```
let a = {};
let b = {}; // два независимых объекта

alert( a == b ); // false
```

Для сравнений типа `obj1 > obj2` или для сравнения с примитивом `obj == 5` объекты преобразуются в примитивы. Мы скоро изучим, как работают такие преобразования объектов, но, по правде говоря, сравнения такого рода необходимы очень редко и обычно являются результатом ошибки программиста.

Клонирование и объединение объектов, Object.assign

Таким образом, при копировании переменной с объектом создаётся ещё одна ссылка на тот же самый объект.

Раздел

Объекты: основы

Навигация по уроку

Сравнение по ссылке

Клонирование и объединение объектов, Object.assign

Вложенное клонирование

Итого

Комментарии

Поделиться

Редактировать на GitHub

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

Это выполнимо, но немного сложно, так как в JavaScript нет встроенного метода для этого. На самом деле, такая нужда возникает редко. В большинстве случаев нам достаточно копирования по ссылке.

Но если мы действительно этого хотим, то нам нужно создавать новый объект и повторять структуру дублируемого объекта, перебирая его свойства и копируя их.

Например так:

```
let user = {
  name: "Иван",
  age: 30
};

let clone = {}; // новый пустой объект

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

// теперь в переменной clone находится абсолютно незави
clone.name = "Пётр"; // изменим в нём данные

alert( user.name ); // в оригинальном объекте значение
```

Кроме того, для этих целей мы можем использовать метод Object.assign.

Синтаксис:

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

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

Например, объединим несколько объектов в один:

```
let user = { name: "Иван" };

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

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

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

Если принимающий объект (`user`) уже имеет свойство с таким именем, оно будет перезаписано:


```
let user = { name: "Иван" };

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

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

Раздел

Объекты: основы

Навигация по уроку

Сравнение по ссылке

Клонирование и объединение объектов, Object.assign

Вложенное клонирование

Итого

Комментарии

Поделиться

Мы также можем использовать `Object.assign` для замены `for..in` на простое клонирование:

```
let user = {
  name: "Иван",
  age: 30
};

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

Этот метод скопирует все свойства объекта `user` в пустой объект и возвратит его.

Вложенное клонирование

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

Например, есть объект:

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

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

Теперь при клонировании недостаточно просто скопировать `clone.sizes = user.sizes`, поскольку `user.sizes` – это объект, он будет скопирован по ссылке. А значит объекты `clone` и `user` в своих свойствах `sizes` будут ссылаться на один и тот же объект:

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

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

alert( user.sizes === clone.sizes ); // true, один и то

// user и clone обращаются к одному sizes
user.sizes.width++;       // меняем свойство в одном об
alert(clone.sizes.width); // 51, видим результат в друг
```

Чтобы исправить это, мы должны в цикле клонирования делать проверку, не является ли значение `user[key]` объектом, и если это так – скопировать и его структуру тоже. Это называется «глубокое клонирование».

Мы можем реализовать глубокое клонирование, используя рекурсию. Или, чтобы не изобретать велосипед, использовать готовую реализацию — метод _.cloneDeep(obj) из JavaScript-библиотеки lodash.

Итого

Объекты присваиваются и копируются по ссылке. Другими словами, переменная хранит не «значение объекта», а «ссылку» (адрес в памяти) на это значение. Поэтому копирование такой переменной или передача её в качестве аргумента функции приводит к копированию этой ссылки, а не самого объекта.

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

Для «простого клонирования» объекта можно использовать `Object.assign`. Необходимо помнить, что `Object.assign` не делает глубокое клонирования объекта. Если внутри копируемого объекта есть свойство значение, которого не является примитивом, оно будет передано по ссылке. Для создания «настоящей копии» (полного клона объекта) можно воспользоваться методом из сторонней JavaScript-библиотеки _.cloneDeep(obj).

Раздел

Объекты: основы

Навигация по уроку

Сравнение по ссылке

Клонирование и объединение объектов, Object.assign

Вложенное клонирование

Итого

Комментарии

Поделиться

Комментарии

перед тем как писать...