

ارجاع به object ها و کپی آنها

@alithecodeguy

یکی از تفاوت‌های اساسی بین object ها و primitive ها این است که primitive ها از هر نوعی که باشند با مقدار کپی می‌شوند یعنی هنگام کپی کردن ، یک کپی از مقدار درون آنها ایجاد می‌شود در صورتی که هنگام کپی کردن object ها ، مقدار درون آنها کپی نمی‌شود بلکه آدرس جایی که آن مقدار ذخیره شده است ، کپی می‌شود.

اگر کمی تیزبینانه‌تر به فرآیند کپی کردن نگاه کنیم ، درک مطلب فوق آسان‌تر می‌گردد.

بیا باید بحث خود را با یک primitive مثل string ادامه دهیم.

در مثال زیر یک کپی از message درون phrase قرار می‌گیرد.

```
let message = "Hello!";
```

```
let phrase = message;
```

در نتیجه ، دو متغیر مستقل از یکدیگر داریم که هر کدام مقدار hello! را درون خودشان دارند.

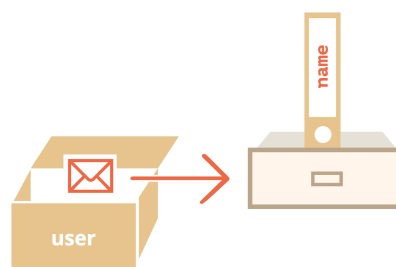


کاملاً واضح است نه؟ object ها اینگونه نیستند 😭

وقتی یک object را به یک متغیر انتساب می‌دهیم ، متغیر حاوی محتویات درون object نیست بلکه حاوی آدرس خانه‌ای از حافظه است که object در آن قرار دارد. در واقع می‌توان گفت که reference ای به آن object است. مثال :

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

طریق عملکرد آن را می‌توان به شکل زیر تصور نمود :



@alithecodeguy

خود object درون جایی در حافظه قرار دارد در صورتی که متغیر user حاوی reference ای به آن خانه حافظه است.

به متغیر user ، یک object variable نیز می گویند .

می توانیم به متغیر آرجکت user به شکل یک کاغذ نگاه کنیم که آدرس object روی آن نوشته شده است .

وقتی کاری با object انجام می دهیم ، مثلاً هنگامی که میخواهیم به مقدار user.name دسترسی داشته باشیم ، جاوا اسکریپت بررسی می کند که در آدرس مورد نظر چه چیزی قرار دارد و عملیات مورد نظر را روی object واقعی انجام می دهد . حال نکته مهمی که وجود دارد این است که :

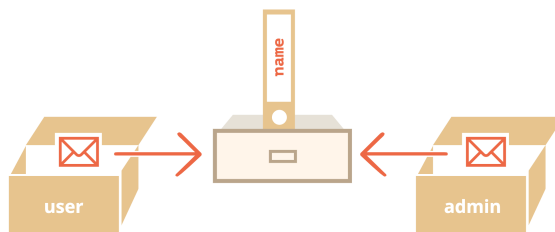
هنگامی که یک object را کپی می کنید ، ارجاع (Reference) کپی می شود ولی خود object اصلی کپی نمی شود .

برای مثال :

```
let user = { name: "John" };  
let admin = user; // copy the reference
```

حال ما دو متغیر داریم که هر دو ارجاعی به یک object یکسان را ذخیره کرده اند :

@alithecodeguy



همانطور که می بینید object یکی است ولی دو ارجاع به آن وجود دارد .

از هر کدام از متغیرها برای دسترسی به object می توانیم استفاده کنیم :

```
let user = { name: 'John' };  
let admin = user;  
admin.name = 'Pete'; // changed by the "admin" reference  
alert(user.name); // 'Pete', changes are seen from the "user" reference
```

مانند این است که یک قفسه با دو کلید برای باز کردن آن داشته باشیم . ابتدا با یک کلید قفسه را باز می کنیم و تغییرات دخواه را انجام می دهیم و در آینده نیز می توانیم با کلید دیگر همان قفسه را باز کرده و تغییرات انجام شده رو ببینیم .

مقایسه با Reference

دو object تنها در صورتی برابرند که هر دو ، یک object باشند . برای مثال در کد زیر متغیر a و متغیر b هر دو به یک object اشاره می کنند . پس هر دو برابرند :

```
let a = {};  
let b = a; // copy the reference  
alert( a == b ); // true, both variables reference the same object  
alert( a === b ); // true
```

ولی در مثال زیر ، چون دو متغیر دو object مجزا هستند ، برابر نیستند حتی اگر به نظر محتویات یکسانی داشته باشند .

```
let a = {};  
let b = {}; // two independent objects  
alert( a == b ); // false
```

برای مقایسه‌هایی مانند obj1>obj2 یا مقایسه object و primitive مانند 5 == obj ، آبجکت‌ها به primitive تبدیل می‌شوند . به زودی در مورد مقایسه object ها توضیح خواهیم داد ولی اگر بخواهیم صادق باشیم ، چنین مقایسه‌ای بسیار نادر بوده و به خطاهای برنامه نویسی منجر می‌گردد .

@alithecodeguy

clone و merge با Object.assign

تا اینجا می‌دانیم که کپی کردن یک object variable منجر به ایجاد یک reference جدید به object یکسان می‌شود . ولی اگر بخواهیم یک object را duplicate کنیم چگونه؟ یک کپی مستقل بسازیم یا یک clone ؟ این کار نیز شدنی است ولی دقت بیشتری را می‌طلبد چرا که متد built-in ای برای این کار در جاوا اسکریپت وجود ندارد . duplicate کردن object ها به ندرت اتفاق می‌افتد و همان کپی با مرجع در اکثر اوقات کفایت می‌کند .

ولی اگر واقعا نیاز داشته باشیم که همچنین کاری انجام دهیم ، باید یک object جدید ساخته و با iterate کردن روی object قبلی ، تمام ساختار آن را روی object جدید نیز ایجاد کنیم سپس مقادیر property ها را به شکل primitive کپی کنیم . مثال :

```
let user = {  
  name: "John",  
  age: 30  
};  
let clone = {}; // the new empty object  
// let's copy all user properties into it  
for (let key in user) {
```

@alithecodeguy

```
clone[key] = user[key];
}
// now clone is a fully independent object with the same content
clone.name = "Pete"; // changed the data in it
alert( user.name ); // still John in the original object
```

همچنین از متد `Object.assign` هم می‌توانیم برای این منظور استفاده کنیم. گرامر آن بدین شکل است :

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

– اولین آرگومان `dest`، آبجکتی است که می‌خواهیم بسازیم.

– باقی آرگومانها مانند `src1 , ... , srcN` (هر تعداد دلخواهی می‌توانند باشند) ، آبجکت‌های اولیه ما هستند.

– در گرامر فوق ، `property` های تمامی آبجکت‌های اولیه `src1 , ... , srcN` ، در آبجکت `dest` کپی می‌شوند. به عبارت دیگر ، `property` های آرگومانهای دوم الی آخر ، در اولین `object` کپی می‌شوند.

– در نهایت آبجکت `dest` برگردانده می‌شود.

برای مثال می‌توانیم از آن برای ترکیب چندین `object` با یکدیگر استفاده کنیم:

```
let user = { name: "John" };
let permissions1 = { canView: true };
let permissions2 = { canEdit: true };
// copies all properties from permissions1 and permissions2 into user
Object.assign(user, permissions1, permissions2);
// now user = { name: "John", canView: true, canEdit: true }
```

تمام `property name` هایی که از قبل وجود داشته‌اند نیز کپی می‌شوند و این یعنی آنها `overwrite` خواهند شد.

```
let user = { name: "John" };
Object.assign(user, { name: "Pete" });
alert(user.name); // now user = { name: "Pete" }
```

همچنین برای `clone` های ساده می‌توانیم به جای `for...in` از `Object.assign` نیز استفاده کنیم :

```
let user = {
  name: "John",
  age: 30
};
let clone = Object.assign({}, user);
```

به این شکل تمام `property` های `user` داخل یک `object` خالی کپی شده و آن `object` برگردانده می‌شود.

تا اینجا فرض کردیم که همه valueهای propertyهای user از نوع primitive هستند. ولی propertyها نیز می توانند اشاره گرهایی به سایر objectها باشند. با آنها چکار کنیم؟ مثال زیر را در نظر بگیرید :

```
let user = {  
  name: "John",  
  sizes: {  
    height: 182,  
    width: 50  
  }  
};  
alert( user.sizes.height ); // 182
```

در این حالت کپی کردن `clone.sizes = user.sizes` کفایت نمی کند چرا که `user.sizes` خودش یک object است و با reference کپی می شود. پس `clone` و `user` به object یکسانی اشاره می کنند. مثال :

```
let user = {  
  name: "John",  
  sizes: {  
    height: 182,  
    width: 50  
  }  
};  
let clone = Object.assign({}, user);  
alert( user.sizes === clone.sizes ); // true, same object  
// user and clone share sizes  
user.sizes.width++; // change a property from one place  
alert(clone.sizes.width); // 51, see the result from the other one
```

برای حل این مشکل باید object های درونی را نیز iterate کنیم و propertyهای آنها را نیز تک تک کپی کنیم. به این کار deep cloning می گویند.

از توابع بازگشتی نیز می توانیم برای این کار استفاده کنیم ، یا برای اینکه چرخ را دوباره اختراع نکنیم می توانیم از پیاده سازی های آماده استفاده کنیم مانند متد `_.cloneDeep(obj)` در کتابخانه lodash



Object های تعریف شده با const می توانند تغییر کنند

یکی از مهمترین اثرات جانبی ذخیره اطلاعات در object ها این است که object ای که با const ، declare شود می تواند تغییر کند.

مثال :

```
const user = {  
  name: "John"  
};  
user.name = "Pete"; // (*)  
alert(user.name); // Pete
```

@alithecodeguy

شاید اینگونه به نظر برسد که خط مشخص شده با * خطا خواهد داد ولی اینطور نیست. user با const ساخته شده است پس همیشه باید به object یکسانی اشاره کند ولی property های آن می توانند آزادانه تغییر کنند. به عبارت دیگر ، const user تنها در صورتی خطا خواهد داد که عملیاتی مانند ... user = انجام دهیم و بخواهیم تمام مقدار آن را تغییر دهیم. اگر بخواهیم object هایی ایجاد کنیم که واقعا const بوده و تغییر نکنند باید از متدهای دیگری استفاده کنیم. در مورد این متدها در بخش های آینده صحبت خواهیم نمود. (بخش Property flags and descriptors)

خلاصه

object ها با مرجع ، مقداردی و کپی می شوند. به عبارت دیگر object متغیری است که object value را ذخیره نمی کند بلکه آدرس آن در memory را ذخیره می کند. پس کپی کردن یا پاس دادن همچنین object ای به فانکشن ها ، آن آدرس یا مرجع را کپی کرده و خود object را کپی نمی کند.

هر عملیاتی روی کپی های مختلف از یک object (مانند حذف و اضافه) ، روی object یکسانی اتفاق می افتد. برای کپی واقعی (clone) ، می توانیم از Object.assign استفاده کنیم که اگر object های داخلی با مرجع کپی شوند به آن shallow copy گفته و اگر همه property ها در object های درونی به صورت تک تک کپی شوند به آن deep cloning می گویند و آن را بوسیله متدهایی مانند cloneDeep(obj) از کتابخانه lodash انجام می دهند.

@alithecodeguy