ارجاع به objectها و کپی آنها

یکی از تفاوتهای اساسی بین objectها و primitiveها این است که primitiveها 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 objectها ، 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 می گردد .

بیاید بحث خود را با یک primitive مثل string ادامه دهیم .

در مثال زیر یک کپی از message درون phrase قرار می گیرد .

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

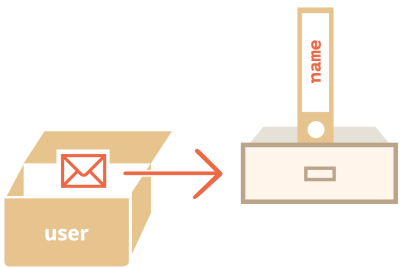
در نتیجه ، دو متغیر مستقل از یکدیگر داریم که هرکدام مقدار hello! را درون خودشان دارند .

کاملا واضح است نه؟ objectها اینگونه نیستند 😂

وقتی یک object را بـه یک متغیر انـتساب می‌دهیم ، متغیر حـاوی مـحتویات درون object نیست بلکه حـاوی آدرس خـانـه‌ای از حافظه است که object در آن قرار دارد . در واقع می توان گفت که refrenceای به آن object است . مثال :

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

طریق عملکرد آن را می‌توان به شکل زیر تصور نمود :



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

به متغیر user ، یک object variable نیز می گویند .

می‌توانیم به متغیر آبجکت user به شکل یک کاغذ نگاه کنیم که آدرس object روی آن نوشته شده است .

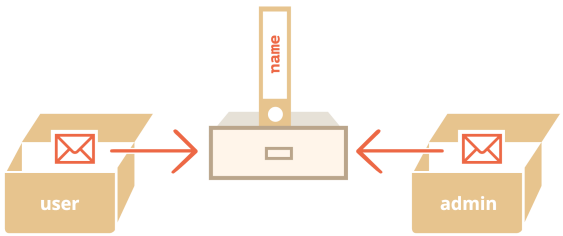
وقتی کاری با object انجام می‌دهیم ، مثلا هنگامی که میخواهیم به مقدار user.name دسترسی داشته باشیم ، جاوا اسکریپت بررسی می‌کند که در آدرس مورد نظر چه چیزی قرار دارد و عملیات مورد نظر را روی object واقعی انجام می‌دهد .حال نکته مهمی که وجود دارد این است که :

هنگامی که یک object را کپی می‌کنید ، ارجاع (Reference) کپی می‌شود ولی خود object اصلی کپی نمی‌شود .

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 کرده‌اند :



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

از هر کدام از متغیرها برای دسترسی به object می‌توانیم 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 می‌کنیم و تغییرت دلخواه را انجام می‌دهیم و در آینده نیز می‌توانیم 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 می‌کنند . 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 obj == 5 ، آبجکت‌ها به primitive تبدیل می‌شوند . به زودی در مورد مقایسه objectها توضیح خواهیم داد ولی اگر بخواهیم صادق باشیم ، چنین مقایسه‌ای بسیار نادر بوده و به خطاهای برنامه نویسی منجر می گردد .

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 چطور؟ 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 objectها به ندرت اتفاق می‌افتد و همان کپی با مرجع در اکثر اوقات کفایت می‌کند .

ولی اگر واقعا نیاز داشته باشیم که همچین 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) {
```

```
  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 ، آبجتی است که می‌خواهیم بسازیم .
- باقی آرگومانها مانند src1 , ... , srcN (هر تعداد دلخواهی می‌توانند باشند) ، آبجکت‌های اولیه ما هستند .
- در گرامر فوق ، propertyهای تمامی آبجکت‌های اولیه src1 , ... , srcN ، در آبجکت dest کپی می‌شوند . به عبارت دیگر ، propertyهای 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 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 cloneهای 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 propertyهای 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 می‌توانند اشاره‌گرهایی 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 نمی‌کند چرا که user.sizes خودش یک object است و با refrence کپی می‌شود . پس clone و user به object یکسانی اشاره می‌کنند . 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 می‌گویند .

از توابع بازگشتی نیز می‌توانیم برای این کار استفاده کنیم ، یا برای اینکه چرخ را دوباره اختراع نکنیم می‌توانیم از پیاده‌سازی‌های آماده استفاده کنیم مانند متد _.cloneDeep(obj) در کتابخانه lodash

Objectهای تعریف شده با const می‌توانند تغییر کنند 😐

یکی از مهمترین اثرات جانبی ذخیره اطلاعات در objectها این است که objectای که با const ، declare شود می‌تواند تغییر کند .

مثال :

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


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 propertyهای آن می‌توانند 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 objectهایی ایجاد کنیم که واقعا const بوده و تغییر نکنند باید از متدهای دیگری استفاده کنیم . در مورد این متدها در بخش‌های آینده صحبت خواهیم نمود . (بخش Property flags and descriptors)

خلاصه

object ها با مرجع ، مقداردهی و کپی می‌شوند . به عبارت دیگر object متغیری است که object value را ذخیره نمی‌کند بلکه آدرس آن در memory را ذخیره می‌کند . پس کپی کردن یا پاس دادن همچین objectای به فانکشن‌ها ، آن آدرس یا مرجع را کپی کرده و خود object را کپی نمی‌کند .
هر عملیاتی روی کپی‌های مختلف از یک object (مانند حذف و اضافه) ، روی object یکسانی اتفاق می‌افتد .
برای کپی واقعی (clone) ، می‌توانیم از Object.assign استفاده کنیم که اگر objectهای داخلی با مرجع کپی شوند به آن shallow copy گفته واگر همه propertyها در objectهای درونی به صورت تک تک کپی شوند به آن deep cloning می‌گویند و آن را بوسیله متدهایی مانند cloneDeep(obj). از کتابخانه lodash انجام می‌دهند .