

Object ها

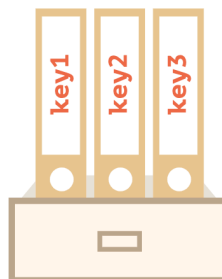
@alithecodeguy

همانطور که از قسمت "انواع داده‌ها" می‌دانیم ، ۸ نوع دیتا در جاوا اسکریپت وجود دارد . به هفت تا از آنها primitive می‌گویند چرا که مقدار آنها فقط شامل یک چیز می‌شود . (یک عدد ، یک رشته یا هر چیز دیگری)

در مقابل ، از object ها برای ذخیره مجموعه‌های کلید دار از انواع مختلف و ماهیت‌های پیچیده استفاده می‌شود . در جاوا اسکریپت مفهوم object تقریباً همه‌جا وجود دارد . پس قبل از هرچیز باید به خوبی با object آشنا بشویم .

یک object می‌تواند بوسیله آکولاد {...} به همراه لیستی دلخواه از مشخصات ساخته شود . یک مشخصه (property) ، یک جفت کلید: مقدار (key:value) بوده که key از نوع رشته‌ای و value از هر نوعی می‌تواند باشد . به key ، property ، name نیز می‌گویند .

یک object را می‌توانیم به شکل یک قفسه همراه با فایل‌های امضا شده در نظر بگیریم . هر بخش از دیتا درون فایل متناظر با key خودش ذخیره می‌گردد . در این حالت پیدا کردن یک فایل بوسیله اسم آن یا اضافه یا حذف یک فایل آسان می‌باشد .



به ۲ طریق زیر می‌توان یک object خالی ایجاد نمود :

```
let user = new Object(); // "object constructor" syntax
```

```
let user = {}; // "object literal" syntax
```

@alithecodeguy



معمولاً از روش دوم یعنی {...} استفاده می‌شود .

به این روش ایجاد یک object خالی ، object literal می‌گویند .

Literal ها و Property ها

می‌توانیم در هنگام ساخت یک object جدید ، propertyهایی را در قالب جفت‌های key:value درون آن قرار دهیم:

```
let user = { // an object
  name: "John", // by key "name" store value "John"
  age: 30 // by key "age" store value 30
};
```

یک property ، یک کلید (name یا identifier) قبل از علامت : و یک مقدار بعد از آن دارد.

در آبجکت user ، دو property وجود دارد :

۱ - property اول با کلید "name" و مقدار "John"

۲ - property دوم با کلید "age" و مقدار 30

آبجکت نهایی user را می‌توان به شکل قفسه‌ای با دو فایل "name" و "age" در نظر گرفت که می‌توانیم هر زمان که بخواهیم فایل‌های را از آن خوانده ، فایل را به آن اضافه و یا از آن حذف کنیم.

بوسیله علامت نقطه (dot notation) می‌توان به مقادیر propertyها دسترسی داشت.

// get property values of the object:

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

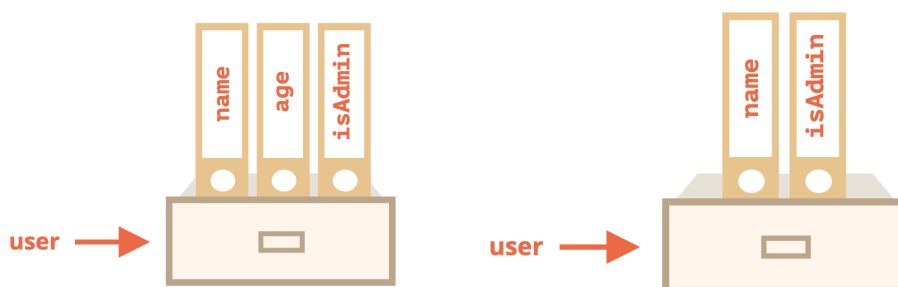
```
alert( user.age ); // 30
```

مقادیر می‌توانند از هر نوعی باشند . بیایید یک مقدار boolean اضافه کنیم:

```
user.isAdmin = true;
```

برای حذف یک property از delete استفاده می‌کنیم:

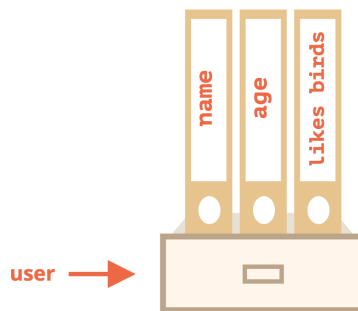
```
delete user.age;
```



پیشاپیش از هرگونه حمایت شما از جمله subscribe در کانال یوتیوب و کانال تلگرام سپاسگزارم.

همچنین می‌توانیم برای نام‌گذاری کلیدها از کلمه‌های چند بخشی استفاده کنیم ولی در این حالت ، اسامی باید بین کوتیشن قرار بگیرند :

```
let user = {  
  name: "John",  
  age: 30,  
  "likes birds": true // multiword property name must be quoted  
};
```



آخرین property در لیست باید با کاما تمام شود :

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

به این کاما ، trailing یا hanging می‌گویند .

این کاما ، اضافه ، حذف و جابه‌جایی خطوط را راحت‌تر می‌کند چرا که همه property ها شبیه هم می‌شوند .

@alithecodeguy

براکت‌ها (square brackets)

برای property هایی که کلید چند بخشی دارند ، روش dot notation کار نمی‌کند :

// this would give a syntax error

```
user.likes birds = true
```

جاوا اسکریپت عبارت بالا را متوجه نمی‌شود و اجرای عبارت بالا به خطا منجر می‌شود .

برای استفاده از dot notation باید کلید انتخاب شده یک نام (identifier) معتبر باشد و این یعنی نام انتخاب شده شامل فضای

خالی نباشد ، با رقم شروع نشده و حاوی کاراکترهای غیر الفبایی و غیر ارقام نباشد . (به جز \$ و _)

برای حل این مشکل روش دیگری وجود دارد به نام square bracket notation که با هر رشته‌ای کار می‌کند :

```
let user = {};  
// set  
user["likes birds"] = true;  
// get  
alert(user["likes birds"]); // true  
// delete  
delete user["likes birds"];
```

حال همه چیز به درستی کار می‌کند. توجه کنید که عبارت داخل براکت درون کوتیشن نوشته شده است (از هر نوع کوتیشن می‌توان استفاده کرد). square bracket notation این امکان را نیز در اختیار ما قرار می‌دهد تا از خروجی عبارات و متغیرها به جای رشته مستقیم استفاده کنیم :

```
let key = "likes birds";  
// same as user["likes birds"] = true;  
user[key] = true;
```

در اینجا متغیر key می‌تواند در زمان اجرا محاسبه شده یا با توجه به ورودی کاربر تغییر کند. سپس ما می‌توانیم از همین متغیر برای دسترسی به مقادیر proeprty استفاده کنیم. این روش استفاده، انعطاف زیادی در انجام کارها در اختیار ما قرار می‌دهد. مثال :

```
let user = {  
  name: "John",  
  age: 30  
}  
let key = prompt("What do you want to know about the user?", "name");  
// access by variable  
alert( user[key] ); // John (if enter "name")
```

در چنین وضعیتی‌هایی نمی‌توانیم از dot notation استفاده نماییم.

```
let user = {  
  name: "John",  
  age: 30  
};  
let key = "name";  
alert( user.key ) // undefined
```

propertyهای محاسبه شده

ما می‌توانیم در زمان ساخت یک object از square bracket استفاده کنیم. به این روش computed properties می‌گویند. برای مثال :

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {
  [fruit]: 5, // the name of the property is taken from the variable fruit
};
alert(bag.apple); // 5 if fruit="apple"
```

معنای یک computed property ساده است. [fruit] یعنی نام property باید از fruit گرفته شود. پس اگر کاربر مقدار apple را وارد کند، آبجکت bag خواهد شد: {apple : 5}

کد بالا را به این صورت نیز می‌توان نوشت :

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {};
// take property name from the fruit variable
bag[fruit] = 5;
```

درون square bracket می‌توانیم عبارات پیچیده‌تری نیز بنویسیم :

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5 // bag.appleComputers = 5
};
```

square bracket از dot notation قوی‌تر بوده و این اجازه را می‌دهد که از هر اسم و متغیری برای نام property استفاده کنیم ولی نوشتن آنها کمی دشوار است. پس در اکثر مواقع، هنگامی که نام property معلوم و ساده است از dot notation استفاده می‌کنیم و اگر به چیز پیچیده‌تری نیاز داشته باشیم از square bracket استفاده می‌کنیم.

مختصرسازی propertyها

در واقعیت، معمولاً از متغیرها هم به عنوان نام property و هم به عنوان مقدار property استفاده می‌کنیم. برای مثال :

```
function makeUser(name, age) {
  return {
    name: name,
```

```
age: age,  
  // ...other properties  
};  
}  
let user = makeUser("John", 30);  
alert(user.name); // John
```

در مثال بالا ، نام **property** ها با نام متغیری که به عنوان مقدار آنها در نظر گرفته شده است ، یکسان است . اینگونه استفاده ، بسیار رایج است به طوری که می توان آن را به روش ساده تری نیز نوشت . به جای **name:name** می توانیم فقط بنویسیم **name** . مثال :

```
function makeUser(name, age) {  
  return {  
    name, // same as name: name  
    age, // same as age: age  
    // ...  
  };  
}
```

هم از **property** های مختصر هم از **property** های عادی به صورت همزمان در یک **object** می توانیم استفاده کنیم .

```
let user = {  
  name, // same as name:name  
  age: 30  
};
```

محدودیت های نام یک property

تا اینجا می دانیم که برای یک متغیر نمی توانیم از نام های رزرو شده استفاده مانند **for** ، **let** ، **return** و ... استفاده کنیم ولی برای **property** های یک **object** چنین محدودیتی نداریم :

```
// these properties are all right  
let obj = {  
  for: 1,  
  let: 2,  
  return: 3  
};  
alert( obj.for + obj.let + obj.return ); // 6
```

به صورت خلاصه ، در نام **property** ها هیچ محدودیتی نداریم . هر **string** یا **symbol** ای می توانند باشند . در مورد **symbol** در بخش های بعد توضیح خواهیم داد .

باقی نوعها به صورت خودکار به رشته تبدیل می‌شوند. برای مثال، هنگامی که از عدد 0 برای نام یک property استفاده کنیم، عدد 0 به رشته "0" تبدیل می‌شود:

```
let obj = {  
  0: "test" // same as "0": "test"  
};  
  
// both alerts access the same property (the number 0 is converted to string "0")  
alert( obj["0"] ); // test  
alert( obj[0] ); // test (same property)
```

تنها نکته‌ای که وجود دارد این است که property مخصوصی به نام __proto__ را نمی‌توانید به مقادیر غیر object ای تغییر دهید:

```
let obj = {};  
obj.__proto__ = 5; // assign a number  
alert(obj.__proto__); // [object Object] - the value is an object, didn't work as intended
```

همانطور که در مثال فوق می‌بینیم، انتساب فوق عمل نکرده است. در بخش‌های بعد در مورد ماهیت __proto__ و نحوه حل مشکل فوق بیشتر صحبت خواهیم کرد.

@alithecodeguy

تست موجود بودن با عملگر in

یکی از ویژگی‌های زبان جاوا اسکریپت در مقابل سایر زبان‌ها این است که تقریباً به همه property ها می‌توان دسترسی داشت. اگر property مورد نظر وجود نداشته باشد، خطایی به وجود نخواهد آمد. استفاده از property ای که وجود ندارد فقط مقدار undefined برمی‌گرداند. پس به راحتی می‌توانیم تست کنیم که یک property وجود دارد یک خیر:

```
let user = {};  
  
alert( user.noSuchProperty === undefined ); // true means "no such property"
```

همچنین عملگر مخصوصی نیز برای این کار وجود دارد که گرامر آن به این شکل است:

"key" in object

برای مثال:

```
let user = { name: "John", age: 30 };  
  
alert( "age" in user ); // true, user.age exists  
  
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

@alithecodeguy

توجه داشته باشید که در سمت چپ عملگر `in` معمولا نام `property` آمده که معمولا نیز به شکل یک `string` داخل کوتیشن است. اگر این کوتیشن‌ها را حذف کنیم، به شکل یک متغیر درمی‌آید که باید حاوی نام `property` مورد نظر ما باشد. برای مثال:

```
let user = { age: 30 };
```

```
let key = "age";
```

```
alert( key in user ); // true, property "age" exists
```

چرا همچنین عملگری وجود دارد؟ آیا همان مقایسه با `undefined` کفایت نمی‌کند؟ خب، در اکثر اوقات مقایسه با `undefined` کفایت می‌کند ولی موقعیت خاصی وجود دارد که این روش به شکل صحیح عمل نمی‌کند ولی در چنین موقعی عملگر `in` درست عمل می‌کند. این موقعیت مخصوص هنگامی است که `property` مورد نظر وجود دارد ولی مقدار `undefined` ر ذخیره کرده است:

```
let obj = {
```

```
  test: undefined
```

```
};
```

```
alert( obj.test ); // it's undefined, so - no such property?
```

```
alert( "test" in obj ); // true, the property does exist!
```

در کد فوق، `obj.test` واقعا وجود دارد. پس عملگر `in` بهتر عمل می‌کند. موقعیت‌های اینچنینی به ندرت پیش می‌آید چرا که `undefined` نباید به صورت صریح به متغیری منتسب شود. اکثر اوقات برای مقادیر خالی یا تعریف نشده از `null` استفاده می‌کنیم. پس عملگر `in` برای جلوگیری از این خطای عجیب و نادر وجود دارد.

حلقه `for...in`

برای عبور از روی همه کلیدهای یک `object` می‌توانیم از حلقه مخصوصی به شکل `for...in` استفاده کنیم. این حلقه با حلقه `for` ای که در بخش‌های قبل در مورد آن صحبت کردیم کاملا متفاوت است و گرامر آن به این شکل است:

```
for (key in object) {
```

```
  // executes the body for each key among object properties
```

```
}
```

برای مثال می‌خواهیم همه `property`های آبجکت `user` را نمایش دهیم:

```
let user = {
```

```
  name: "John",
```



```
age: 30,  
isAdmin: true  
};  
for (let key in user) {  
  // keys  
  alert( key ); // name, age, isAdmin  
  // values for the keys  
  alert( user[key] ); // John, 30, true  
}
```

دقت کنید که ساختار **for** به ما اجازه می دهد که متغیرهای حلقه را درون حلقه تعریف کنیم مانند **let key** در مثال فوق .

همچنین در اینجا می توانستیم از نام دیگری به جز **key** برای متغیر حلقه استفاده کنیم . برای مثال می توانستیم بنویسیم :

```
for (let prop in obj)
```

ترتیب property های یک object

آیا **object** ها مرتب هستند؟ به عبارت دیگر ، آیا اگر از حلقه روی یک **object** استفاده کنیم ، آیا **property** ها را به همان ترتیبی که اضافه شده اند برمی گرداند؟ می توانیم مطمئن باشیم؟

جواب کوتاه این است که : **object** ها به روش خاص خودشان مرتب می شوند . **property** هایی از نوع عدد صحیح **sort** می شوند و باقی بر اساس زمان ایجاد مرتب می شوند .

برای مثال ، **object** ای را در نظر بگیرید که شامل کدهای تلفن است :

```
let codes = {  
  "49": "Germany",  
  "41": "Switzerland",  
  "44": "Great Britain",  
  // ...  
  "1": "USA"  
};
```

```
for (let code in codes) {  
  alert(code); // 1, 41, 44, 49  
}
```

این object ممکن است به گونه‌ای استفاده شود که لیستی از گزینه‌ها را در اختیار کاربر قرار دهد. اگر ما در حال طراحی سایتی برای مخاطبان آلمانی هستیم، پس احتمالا می‌خواهیم 49 در ابتدای لیست قرار بگیرد.

ولی اگر کد را اجرا کنیم، چیز کاملاً متفاوتی خواهیم دید: ابتدا USA(1) می‌آید، سپس کد 41 و باقی کدها.

کدهای تلفن به ترتیب صعودی مرتب می‌شوند چرا که اعداد صحیح هستند. پس به بدین صورت مرتب می‌شوند: 1,41,44,49

@alithecodeguy

propertyهایی از نوع عدد صحیح، چه هستند؟

propertyهای صحیح یعنی یک رشته می‌تواند بدون تغییر به عدد تبدیل شود و برعکس. پس رشته "49" یک property صحیح است، چرا که هنگامی که به یک عدد صحیح تبدیل شده و برمی‌گردد، همچنان همان است. ولی رشته‌های "+49" و "1.2" را نمی‌توان property صحیح محسوب کرد:

```
alert( String(Math.trunc(Number("49"))) ); // "49", same, integer property
```

```
alert( String(Math.trunc(Number("+49"))) ); // "49", not same "+49" ⇒ not integer property
```

```
alert( String(Math.trunc(Number("1.2"))) ); // "1", not same "1.2" ⇒ not integer property
```

در ضمن، اگر کلیدها به شکل غیر صحیح باشند، به ترتیب زمان ساخت مرتب می‌شوند. برای مثال:

```
let user = {  
  name: "John",  
  surname: "Smith"  
};  
  
user.age = 25; // add one more  
  
// non-integer properties are listed in the creation order  
  
for (let prop in user) {  
  alert( prop ); // name, surname, age  
}
```

@alithecodeguy

پس برای حل مشکلاتی مثل کدهای تلفن در مثال بالا ، می‌توانیم حقه زده و کدها را به صورت غیر صحیح وارد کنیم. اضافه کردن علامت + قبل از هر کد کفایت می‌کند :

```
let codes = {  
  
  "+49": "Germany",  
  
  "+41": "Switzerland",  
  
  "+44": "Great Britain",  
  
  // ..,  
  
  "+1": "USA"  
  
};  
  
for (let code in codes) {  
  
  alert( +code ); // 49, 41, 44, 1  
  
}
```

@alithecodeguy

خلاصه

objectها آرایه‌هایی با صفات و ویژگی‌های خاص هستند. آنها propertyها را در قالب جفت‌های key:value ذخیره می‌کنند:

– نام propertyها باید رشته یا symbol باشد. (معمولا رشته)

– مقادیر می‌توانند از هر نوعی باشند.

@alithecodeguy

برای دسترسی به یک property می‌توانیم:

– از dot notation استفاده کنیم : obj.property

– از square bracket استفاده کنیم که این اجازه را می‌دهند برای مقدار key از متغیر استفاده کنیم : obj[varName]

باقی عملگرها :

– برای پاک کردن یک property از delete استفاده می‌کنیم : delete obj.prop

– برای چک کردن وجود داشتن یا نداشتن یک key از عملگر in استفاده می‌کنیم : "key" in obj

– برای iterate روی یک object به شکل زیر عمل می‌کنیم : for(let key in obj)

چیزی که در این بخش مطالعه کردیم plain object یا به شکل ساده تر ، Object خوانده می شود .

انواع دیگری از object ها در جاوا اسکریپت وجود دارد :

– Array : برای ذخیره مجموعه داده ها به شکل مرتب

– Date : برای ذخیره اطلاعات تاریخ و زمان

– Error : برای ذخیره اطلاعات مربوط به خطا

– و کلی object دیگر

همه آنها ویژگی های مختص خودشان را دارند که در بخش های بعد بیشتر بررسی می کنیم . گاهی اوقات افراد از الفاظ Array type یا Date type استفاده می کنند ولی در واقع آنها نوع های جداگانه ای نیستند بلکه به نوع object تعلق دارند و به شکل خاصی توسعه پیدا کرده اند .

در جاوا اسکریپت object ها بسیار قوی هستند . در اینجا تنها به توضیح مختصری از آن بسنده کردیم . در بخش های بعد بیشتر در مورد آن صحبت کرده و تمرین خواهیم نمود .

@alithecodeguy