Object.assign()

`Object.assign()` 方法用于将所有可枚举属性的值从一个或多个源对象复制到目标对象。它将返回目标对象。

语法

```
Object.assign(target, ...sources)
```

参数

`target`
目标对象。

`sources`
源对象。

返回值

目标对象。

描述

如果目标对象中的属性具有相同的键，则属性将被源中的属性覆盖。后来的源的属性将类似地覆盖早先的属性。

`Object.assign` 方法只会拷贝源对象自身的并且可枚举的属性到目标对象。该方法使用源对象的 `[[Get]]` 和目标对象的 `[[Set]]`，所以它会调用相关 getter 和 setter。因此，它分配属性，而不仅仅是复制或定义新的属性。如果合并源包含getter，这可能使其不适合将新属性合并到原型中。为了将属性定义（包括其可枚举性）复制到原型，应使用 `Object.getOwnPropertyDescriptor()` 和 `Object.defineProperty()` 。

`String` 类型和 `Symbol` 类型的属性都会被拷贝。

在出现错误的情况下，例如，如果属性不可写，会引发 `TypeError` ，如果在引发错误之前添加了任何属性，则可以更改 `target` 对象。

注意， `Object.assign` 会跳过那些值为 `null` 或 `undefined` 的源对象。

示例

复制一个对象

```
var obj = { a: 1 };
var copy = Object.assign({}, obj);
console.log(copy); // { a: 1 }
```

深拷贝问题

针对深拷贝，需要使用其他方法，因为 `Object.assign()` 拷贝的是属性值。假如源对象的属性值是一个指向对象的引用，它也只拷贝那个引用值。

```
function test() {
  'use strict';

  let obj1 = { a: 0 , b: { c: 0}};
  let obj2 = Object.assign({}, obj1);
  console.log(JSON.stringify(obj2)); // { a: 0, b: { c: 0}}

  obj1.a = 1;
  console.log(JSON.stringify(obj1)); // { a: 1, b: { c: 0}}
  console.log(JSON.stringify(obj2)); // { a: 0, b: { c: 0}}

  obj2.a = 2;
  console.log(JSON.stringify(obj1)); // { a: 1, b: { c: 0}}
  console.log(JSON.stringify(obj2)); // { a: 2, b: { c: 0}}

  obj2.b.c = 3;
  console.log(JSON.stringify(obj1)); // { a: 1, b: { c: 3}}
  console.log(JSON.stringify(obj2)); // { a: 2, b: { c: 3}}

  // Deep Clone
  obj1 = { a: 0 , b: { c: 0}};
  let obj3 = JSON.parse(JSON.stringify(obj1));
  obj1.a = 4;
  obj1.b.c = 4;
  console.log(JSON.stringify(obj3)); // { a: 0, b: { c: 0}}
}
```

```
27
28 test();
```

合并对象

```
1 var o1 = { a: 1 };
2 var o2 = { b: 2 };
3 var o3 = { c: 3 };
4
5 var obj = Object.assign(o1, o2, o3);
6 console.log(obj); // { a: 1, b: 2, c: 3 }
7 console.log(o1);  // { a: 1, b: 2, c: 3 }, 注意目标对象自身也会改变。
```

合并具有相同属性的对象

```
1 var o1 = { a: 1, b: 1, c: 1 };
2 var o2 = { b: 2, c: 2 };
3 var o3 = { c: 3 };
4
5 var obj = Object.assign({}, o1, o2, o3);
6 console.log(obj); // { a: 1, b: 2, c: 3 }
```

属性被后续参数中具有相同属性的其他对象覆盖。

拷贝 symbol 类型的属性

```
1 var o1 = { a: 1 };
2 var o2 = { [Symbol('foo')]: 2 };
3
4 var obj = Object.assign({}, o1, o2);
5 console.log(obj); // { a : 1, [Symbol("foo")]: 2 } (cf. bug 1207182 on F
6 Object.getOwnPropertySymbols(obj); // [Symbol(foo)]
```

继承属性和不可枚举属性是不能拷贝的

```
1 var obj = Object.create({foo: 1}, { // foo 是个继承属性。
2     bar: {
3         value: 2  // bar 是个不可枚举属性。
      },
```

```
    baz: {
        value: 3,
        enumerable: true  // baz 是个自身可枚举属性。
    }
});

var copy = Object.assign({}, obj);
console.log(copy); // { baz: 3 }
```

原始类型会被包装为对象

```
var v1 = "abc";
var v2 = true;
var v3 = 10;
var v4 = Symbol("foo")

var obj = Object.assign({}, v1, null, v2, undefined, v3, v4);
// 原始类型会被包装，null 和 undefined 会被忽略。
// 注意，只有字符串的包装对象才可能有自身可枚举属性。
console.log(obj); // { "0": "a", "1": "b", "2": "c" }
```

异常会打断后续拷贝任务

```
var target = Object.defineProperty({}, "foo", {
    value: 1,
    writable: false
}); // target 的 foo 属性是个只读属性。

Object.assign(target, {bar: 2}, {foo2: 3, foo: 3, foo3: 3}, {baz: 4});
// TypeError: "foo" is read-only
// 注意这个异常是在拷贝第二个源对象的第二个属性时发生的。

console.log(target.bar);  // 2，说明第一个源对象拷贝成功了。
console.log(target.foo2); // 3，说明第二个源对象的第一个属性也拷贝成功了。
console.log(target.foo);  // 1，只读属性不能被覆盖，所以第二个源对象的第二个属
console.log(target.foo3); // undefined，异常之后 assign 方法就退出了，第三个
console.log(target.baz);  // undefined，第三个源对象更是不会被拷贝到的。
```

拷贝访问器

```
var obj = {
  foo: 1,
  get bar() {
    return 2;
  }
};

var copy = Object.assign({}, obj);
// { foo: 1, bar: 2 }
// copy.bar的值来自obj.bar的getter函数的返回值
console.log(copy);

// 下面这个函数会拷贝所有自有属性的属性描述符
function completeAssign(target, ...sources) {
  sources.forEach(source => {
    let descriptors = Object.keys(source).reduce((descriptors, key) => {
      descriptors[key] = Object.getOwnPropertyDescriptor(source, key);
      return descriptors;
    }, {});

    // Object.assign 默认也会拷贝可枚举的Symbols
    Object.getOwnPropertySymbols(source).forEach(sym => {
      let descriptor = Object.getOwnPropertyDescriptor(source, sym);
      if (descriptor.enumerable) {
        descriptors[sym] = descriptor;
      }
    });
    Object.defineProperties(target, descriptors);
  });
  return target;
}

var copy = completeAssign({}, obj);
console.log(copy);
// { foo:1, get bar() { return 2 } }
```

Polyfill

此polyfill不支持 symbol 属性，因为ES5 中根本没有 symbol：

```
if (typeof Object.assign != 'function') {
  // Must be writable: true, enumerable: false, configurable: true
  Object.defineProperty(Object, "assign", {
    value: function assign(target, varArgs) { // .length of function is 2
      'use strict';
      if (target == null) { // TypeError if undefined or null
        throw new TypeError('Cannot convert undefined or null to object');
      }

      var to = Object(target);

      for (var index = 1; index < arguments.length; index++) {
        var nextSource = arguments[index];

        if (nextSource != null) { // Skip over if undefined or null
          for (var nextKey in nextSource) {
            // Avoid bugs when hasOwnProperty is shadowed
            if (Object.prototype.hasOwnProperty.call(nextSource, nextKey)) {
              to[nextKey] = nextSource[nextKey];
            }
          }
        }
      }
      return to;
    },
    writable: true,
    configurable: true
  });
}
```

规范

规范名称	规范状态	备注
ECMAScript 2015 (6th Edition, ECMA-262) Object.assign	ST Standard	Initial definition.
ECMAScript Latest Draft (ECMA-262) Object.assign	LS Living Standard	

浏览器兼容

Desktop Mobile

Feature	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari
Basic Support	45	(Yes)	34	No	32	9

相关链接

- `Object.defineProperties()`
- 属性的可枚举性和所有权