es6之扩展运算符 三个点（...）

对象的扩展运算符

理解对象的扩展运算符其实很简单，只要记住一句话就可以：

> **对象中的扩展运算符(...)用于取出参数对象中的所有可遍历属性，拷贝到当前对象之中**

```
let bar = { a: 1, b: 2 };
let baz = { ...bar }; // { a: 1, b: 2 }
```

上述方法实际上等价于:

```
let bar = { a: 1, b: 2 };
let baz = Object.assign({}, bar); // { a: 1, b: 2 }
```

`Object.assign` 方法用于对象的合并，将源对象 `(source)` 的所有可枚举属性，复制到目标对象 `(target)` 。

`Object.assign` 方法的第一个参数是目标对象，后面的参数都是源对象。(**如果目标对象与源对象有同名属性，或多个源对象有同名属性，则后面的属性会覆盖前面的属性**)。

同样，如果用户自定义的属性，放在扩展运算符后面，则扩展运算符内部的同名属性会被覆盖掉。

```
let bar = {a: 1, b: 2};
let baz = {...bar, ...{a:2, b: 4}};  // {a: 2, b: 4}
```

利用上述特性就可以很方便的修改对象的部分属性。在 `redux` 中的 `reducer` 函数规定必须是**一个纯函数**（如果不是很清楚什么是纯函数的可以参考这里）， `reducer` 中的 `state` 对象要求不能直接修改，可以通过扩展运算符把修改路径的对象都复制一遍，然后产生一个新的对象返回。

这里有点需要注意的是扩展运算符对对象实例的拷贝属于一种浅拷贝。肯定有人要问什么是浅拷贝？我们知道 `javascript` 中有两种数据类型，分别是*基础数据类型*和*引用数据类型*。*基础数据类型*是按值访问的，常见的*基础数据类型*有 `Number` 、 `String` 、 `Boolean` 、 `Null` 、 `Undefined` ，这类变量的拷贝的时候会完整的复制一份；*引用数据类型*比如 `Array` ，在拷贝的时候拷贝的是对象的引用，当原对象发生变化的时候，拷贝对象也跟着变化，比如：

```
let obj1 = { a: 1, b: 2};
let obj2 = { ...obj1, b: '2-edited'};
console.log(obj1); // {a: 1, b: 2}
console.log(obj2); //  {a: 1, b: "2-edited"}
```

上面这个例子扩展运算符拷贝的对象是*基础数据类型*，因此对 `obj2` 的修改并不会影响 `obj1` ，如果改成这样：

```
let obj1 = { a: 1, b: 2, c: {nickName: 'd'}};
let obj2 = { ...obj1};
obj2.c.nickName = 'd-edited';
console.log(obj1); // {a: 1, b: 2, c: {nickName: 'd-edited'}}
console.log(obj2); // {a: 1, b: 2, c: {nickName: 'd-edited'}}
```

这里可以看到，对 `obj2` 的修改影响到了被拷贝对象 `obj1` ，原因上面已经说了，因为 `obj1` 中的对象 `c` 是一个引用数据类型，拷贝的时候拷贝的是对象的引用。

数组的扩展运算符

扩展运算符同样可以运用在对数组的操作中。

- 可以将数组转换为参数序列

```
function add(x, y) {
  return x + y;
}

const numbers = [4, 38];
add(...numbers) // 42
```

- 可以复制数组

如果直接通过下列的方式进行数组复制是不可取的：

```
const arr1 = [1, 2];
const arr2 = arr1;
arr2[0] = 2;
arr1 // [2, 2]
```

原因上面已经介绍过，用扩展运算符就很方便：

```
const arr1 = [1, 2];
const arr2 = [...arr1];
```

还是记住那句话：**扩展运算符(...)用于取出参数对象中的所有可遍历属性，拷贝到当前对象之中**，这里参数对象是个数组，数组里面的所有对象都是基础数据类型，将所有基础数据类型重新拷贝到新的数组中。

- 扩展运算符可以与解构赋值结合起来，用于生成数组

```
const [first, ...rest] = [1, 2, 3, 4, 5];
first // 1
rest  // [2, 3, 4, 5]
```

需要注意的一点是：

> **如果将扩展运算符用于数组赋值，只能放在参数的最后一位，否则会报错。**

```
const [...rest, last] = [1, 2, 3, 4, 5];
// 报错
const [first, ...rest, last] = [1, 2, 3, 4, 5];
// 报错
```

- 扩展运算符还可以将字符串转为真正的数组

```
[...'hello']
// [ "h", "e", "l", "l", "o" ]
```

- 任何 Iterator 接口的对象（参阅 Iterator 一章），都可以用扩展运算符转为真正的数组

这点说的比较官方，大家具体可以参考阮一峰老师的ECMAScript 6入门教程。

比较常见的应用是可以将某些数据结构转为数组,比如：

```
// arguments对象
function foo() {
  const args = [...arguments];
}
```

用于替换 es5 中的 Array.prototype.slice.call(arguments) 写法。