# 你会用ES6，那倒是用啊！

**ps：ES5之后的JS语法统称ES6！！！**

## 一、关于取值的吐槽

取值在程序中非常常见，比如从对象`obj`中取值。

```
const obj = {
    a:1,
    b:2,
    c:3,
    d:4,
    e:5,
}
复制代码
```

**吐槽**：

```
const a = obj.a;
const b = obj.b;
const c = obj.c;
const d = obj.d;
const e = obj.e;
复制代码
```

或者

```
const f = obj.a + obj.d;
const g = obj.c + obj.e;
复制代码
```

吐槽：“不会用ES6的解构赋值来取值吗？5行代码用1行代码搞定不香吗？直接用对象名加属性名去取值，要是对象名短还好，很长呢？搞得代码中到处都是这个对象名。”

**改进**：

```
const {a,b,c,d,e} = obj;
const f = a + d;
const g = c + e;
复制代码
```

**反驳**

不是不用ES6的解构赋值，而是服务端返回的数据对象中的属性名不是我想要的，这样取值，不是还得重新创建个遍历赋值。

**吐槽**

看来你对ES6的解构赋值掌握的还是不够彻底。如果想创建的变量名和对象的属性名不一致，可以这么写：

```
const {a:a1} = obj;
console.log(a1);// 1
复制代码
```

**补充**

ES6的解构赋值虽然好用。但是要注意解构的对象不能为`undefined`、`null`。否则会报错，故要给被解构的对象一个默认值。

```
const {a,b,c,d,e} = obj || {};
复制代码
```

# 二、关于合并数据的吐槽

比如合并两个数组，合并两个对象。

```
const a = [1,2,3];
const b = [1,5,6];
const c = a.concat(b);//[1,2,3,1,5,6]

const obj1 = {
  a:1,
}
const obj1 = {
  b:1,
}
const obj = Object.assgin({}, obj1, obj2);//{a:1,b:1}
复制代码
```

**吐槽**

ES6的扩展运算符是不是忘记了，还有数组的合并不考虑去重吗?

**改进**

```
const a = [1,2,3];
const b = [1,5,6];
const c = [...new Set([...a,...b])];//[1,2,3,5,6]

const obj1 = {
  a:1,
}
const obj2 = {
  b:1,
}
const obj = {...obj1,...obj2};//{a:1,b:1}
复制代码
```

# 三、关于拼接字符串的吐槽

```
const name = '小明';
const score = 59;
const result = '';
if(score > 60){
  result = `${name}的考试成绩及格`;
}else{
  result = `${name}的考试成绩不及格`;
}
复制代码
```

**吐槽**

像你们这样用ES6字符串模板，还不如不用，你们根本不清楚在`${}`中可以做什么操作。在`${}`中可以放入任意的JavaScript表达式，可以进行运算，以及引用对象属性。

**改进**

```
const name = '小明';
const score = 59;
const result = `${name}${score > 60?'的考试成绩及格':'的考试成绩不及格'}`;
复制代码
```

# 四、关于if中判断条件的吐槽

```
if(
    type == 1 ||
    type == 2 ||
    type == 3 ||
    type == 4 ||
){
   //...
}
复制代码
```

**吐槽**

ES6中数组实例方法`includes`会不会使用呢?

**改进**

```
const condition = [1,2,3,4];

if( condition.includes(type) ){
   //...
}
复制代码
```

# 五、关于列表搜索的吐槽

在项目中，一些没分页的列表的搜索功能由前端来实现，搜索一般分为精确搜索和模糊搜索。搜索也要叫过滤，一般用`filter`来实现。

```
const a = [1,2,3,4,5];
const result = a.filter(
  item =>{
    return item === 3
  }
)
```
复制代码

**吐槽**

如果是精确搜索不会用ES6中的`find`吗？性能优化懂么，`find`方法中找到符合条件的项，就不会继续遍历数组。

**改进**

```
const a = [1,2,3,4,5];
const result = a.find(
  item =>{
    return item === 3
  }
)
```
复制代码

# 六、关于扁平化数组的吐槽

一个部门JSON数据中，属性名是部门id，属性值是个部门成员id数组集合，现在要把有部门的成员id都提取到一个数组集合中。

```
const deps = {
'采购部':[1,2,3],
'人事部':[5,8,12],
'行政部':[5,14,79],
'运输部':[3,64,105],
}
let member = [];
for (let item in deps){
    const value = deps[item];
    if(Array.isArray(value)){
        member = [...member,...value]
    }
}
member = [...new Set(member)]
```
复制代码

**吐槽**

获取对象的全部属性值还要遍历吗？`Object.values`忘记了吗？还有涉及到数组的扁平化处理，为啥不用ES6提供的`flat`方法呢，还好这次的数组的深度最多只到2维，还要是遇到4维、5维深度的数组，是不是得循环嵌套循环来扁平化？

**改进**

```
const deps = {
    '采购部':[1,2,3],
    '人事部':[5,8,12],
    '行政部':[5,14,79],
    '运输部':[3,64,105],
}
let member = Object.values(deps).flat(Infinity);
复制代码
```

其中使用`Infinity`作为`flat`的参数，使得无需知道被扁平化的数组的维度。

**补充**

`flat`方法不支持IE浏览器。

# 七、关于获取对象属性值的吐槽

```
const name = obj && obj.name;
复制代码
```

**吐槽**

ES6中的可选链操作符会使用么？

**改进**

```
const name = obj?.name;
复制代码
```

# 八、关于添加对象属性的吐槽

当给对象添加属性时，如果属性名是动态变化的，该怎么处理。

```
let obj = {};
let index = 1;
let key = `topic${index}`;
obj[key] = '话题内容';
复制代码
```

**吐槽**

为何要额外创建一个变量。不知道ES6中的对象属性名是可以用表达式吗？

**改进**

```
let obj = {};
let index = 1;
obj[`topic${index}`] = '话题内容';
复制代码
```

# 九、关于输入框非空的判断

在处理输入框相关业务时，往往会判断输入框未输入值的场景。

```
if(value !== null && value !== undefined && value !== ''){
    //...
}
```
复制代码

**吐槽**

ES6中新出的空值合并运算符了解过吗，要写那么多条件吗？

```
if(value??'' !== ''){
  //...
}
```
复制代码

# 十、关于异步函数的吐槽

异步函数很常见，经常是用 Promise 来实现。

```
const fn1 = () =>{
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(1);
    }, 300);
  });
}
const fn2 = () =>{
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(2);
    }, 600);
  });
}
const fn = () =>{
   fn1().then(res1 =>{
      console.log(res1);// 1
      fn2().then(res2 =>{
        console.log(res2)
      })
   })
}
```
复制代码

**吐槽**

如果这样调用异步函数，不怕形成地狱回调啊！

**改进**

```
const fn = async () =>{
  const res1 = await fn1();
  const res2 = await fn2();
  console.log(res1);// 1
  console.log(res2);// 2
}
```
复制代码

**补充**

但是要做并发请求时，还是要用到`Promise.all()`。

```
const fn = () =>{
   Promise.all([fn1(),fn2()]).then(res =>{
       console.log(res);// [1,2]
   })
}
复制代码
```

如果并发请求时，只要其中一个异步函数处理完成，就返回结果，要用到`Promise.race()`。