- **可选链操作符**

```
if (data && data.list && data.list.name) {
    //do something
}
// 我们都知道这么写是为了容错，减少代码异常问题，但是这么写很不优雅


if (data ? .list ? .name) {
    // do something
}
// 怎么样是不是优雅多了
// 如果list不存在（null/undefined）的话，表达式就会直接返回undefined

```

- **空位合并操作符**

```
let c = a ? a : b
// 或者let c = a || b
// 意思不用我解释了吧，a存在话将a赋给c，a不存在的话将b赋给c


let c = a || b 弊端
string: "" || v
boolean: false || v
nubmer: 0 || v
number: NaN || v
当第一个值为0 false NAN  '' 等情况的时候，也会将b赋给c，但有时候0 或者false可能是个有效值


let c = a ?? b
// 仅在第一项为null或者undefined时，才会使用第二个值
```

- **空值合并运算符** 与 **可选链** 相结合，可以很轻松处理多级查询并赋予默认值问题。

```
var level = user.data?.level ?? '暂无等级';
```

- **Object.entries()** 和 **Object.fromEntries()**

```
let obj = {
  a:1,
  b:2,
  c:3
}
console.log(Object.entries(obj))
// [ [ 'a', 1 ], [ 'b', 2 ], [ 'c', 3 ] ]
for (let [key, value] of Object.entries(obj)) {
  console.log(key, value)
}
// a 1
// b 2
// c 3

let arr = [['a', 1], ['b', 2], ['c', 3]]
console.log(Object.fromEntries(arr))
```

```
// { a: 1, b: 2, c: 3 }

```

- **Promise.allSettled()**

```
const resolved = Promise.resolve(42);
const rejected = Promise.reject(-1);

const allSettledPromise = Promise.allSettled([resolved, rejected]);

allSettledPromise.then(function (results) {
  console.log(results);
});
// [
//    { status: 'fulfilled', value: 42 },
//    { status: 'rejected', reason: -1 }
// ]

```