22 代码重构

本书目的不是讲解重构，但其实目前为止都在进行代码级别的优化，通过反例代码讲解设计模式重构之后的代码

模式与重构是与生俱来的关系，设计模式就是为了重构行为提供目标

实际项目中除了设计模式，还有一些容易忽略细节帮助我们达到重构目的，这里我们介绍一部分

提炼函数

JS中，函数体内最好逻辑清晰明了，过长的话看是否可以独立一部分逻辑：

- 避免出现超大函数
- 独立出来的容易复用
- 独立出来的函数有良好的命名，充当注释

合并重复的条件片段

如果一个函数体内有条件分支语句，并且内部散步了重复的代码，我们是否可以进行合并去重工作来优化。例如如下进行currPage边界保护的跳转：

```
const paging = function(currPage){
    if(currPage <= 0){
        currPage = 0;
        jump(currPage)
    } else if(currPage >= totalPage){
        currPage = totalPage;
        jump(currPage)
    } else {
        jump(currPage)
    }
}
```

这里我们完全可以将重复的jump抽离出来：

```
const paging = function(currPage){
    if(currPage <= 0){
        currPage = 0;
```

```
    } else if(currPage >= totalPage){
        currPage = totalPage;
    }
    jump(currPage)
}
```

复杂的分值语句提炼成函数

这里如有一个规则：如果是夏天，所有商品8折出售，这里我们就可以把判断是夏天的逻辑抽离处来，一方面代码意图明显，另一方面代替了注释：

```
const isSummer = function(){
    const date = new Date();
    return date.getMonth() >= 6 *&& date.getMonth() <= 9
}

const getPrice = function(price){
    if(isSummer){
        return price * 0.8
    }
    return price
}
```

提前让函数推出代替嵌套条件分支

许多程序员都有一个观念，函数只有一个入口和出口，但是只有一个出口其实有不同看法，有时候嵌套的if-else复杂且冗长之后之后会影响可读性，这个时候其实可以挑选一些条件分支，例如把外层的if反转来重构：

```
const del = function(obj){
    if(obj.isReadOnly){
        return;
    }
    if(obj.isFolder){
        return deleteFolder(obj)
    }
}
```