💩一样的代码，你写过没有啊

准则

准则

💩 以一种代码已经被混淆的方式命名变量

如果我们键入的东西越少，那么就有越多的时间去思考代码逻辑等问题。

Good 👍🏻

```
let a = 42;
```

Bad 👎🏻

```
let age = 42;
```

💩 变量/函数混合命名风格

为不同庆祝一下。

Good 👍

```
let wWidth = 640;
let w_height = 480;
```

Bad 👎

```
let windowWidth = 640;
let windowHeight = 480;
```

💩 不要写注释

反正没人会读你的代码。

Good 👍

```
const cdr = 700;
```

Bad 👎

更多时候，评论应该包含一些“为什么”，而不是一些“是什么”。如果“什么”在代码中不清楚，那么代码可能太混乱了。

```
// 700ms的数量是根据UX A/B测试结果进行经验计算的。
// @查看: <详细解释700的一个链接>
const callbackDebounceRate = 700;
```

💩 使用母语写注释

如果您违反了“无注释”原则，那么至少尝试用一种不同于您用来编写代码的语言来编写注释。如果你的母语是英语，你可能会违反这个原则。

Good 👍

```
// Закриваємо модальне віконечко при виникненні помилки.
toggleModal(false);
```

Bad 👎

```
// 隐藏错误弹窗
toggleModal(false);
```

💩 尽可能混合不同的格式

为不同庆祝一下。

Good 👍

```
let i = ['tomato', 'onion', 'mushrooms'];
let d = [ "ketchup", "mayonnaise" ];
```

Bad 👎

```
let ingredients = ['tomato', 'onion', 'mushrooms'];
let dressings = ['ketchup', 'mayonnaise'];
```

💩 尽可能把代码写成一行

Good 👍🏻

```
document.location.search.replace(/(^\?)/,'').split('&').reduce(function(o,n)
{n=n.split('=');o[n[0]]=n[1];return o},{})
```

Bad 👎🏻

```
document.location.search
  .replace(/(^\?)/, '')
  .split('&')
  .reduce((searchParams, keyValuePair) => {
    keyValuePair = keyValuePair.split('=');
    searchParams[keyValuePair[0]] = keyValuePair[1];
    return searchParams;
  },
  {}
)
```

💩 不要处理错误

无论何时发现错误，都没有必要让任何人知道它。没有日志，没有错误弹框。

Good 👍🏻

```
try {
  // 意料之外的情况。
} catch (error) {
  // tss... 🤫
}
```

Bad 👎🏻

```
try {
  // 意料之外的情况。
} catch (error) {
  setErrorMessage(error.message);
  // and/or
  logError(error);
}
```

💩 广泛使用全局变量

全球化的原则。

Good 👍🏻

```
let x = 5;

function square() {
  x = x ** 2;
}
```

```
square(); // 现在x是25
```

Bad 👎

```
let x = 5;

function square(num) {
  return num ** 2;
}

x = square(x); // 现在x是25
```

💩 创建你不会使用的变量

以防万一。

Good 👍

```
function sum(a, b, c) {
  const timeout = 1300;
  const result = a + b;
  return a + b;
}
```

Bad 👎

```
function sum(a, b) {
  return a + b;
}
```

💩 如果语言允许，不要指定类型和/或不执行类型检查。

Good 👍

```
function sum(a, b) {
  return a + b;
}

// 在这里享受没有注释的快乐
const guessWhat = sum([], {}); // -> "[object Object]"
const guessWhatAgain = sum({}, []); // -> 0
```

Bad 👎

```
function sum(a: number, b: number): ?number {
  // 当我们在JS中不做置换和/或流类型检查时，覆盖这种情况。
  if (typeof a !== 'number' && typeof b !== 'number') {
    return undefined;
  }
  return a + b;
```

```
}

// 这个应该在转换/编译期间失败。
const guessWhat = sum([], {}); // -> undefined
```

💩 你应该有不能到达的代码

这是你的 "Plan B".

Good 👍🏻

```
function square(num) {
  if (typeof num === 'undefined') {
    return undefined;
  }
  else {
    return num ** 2;
  }
  return null; // 这就是我的"Plan B".
}
```

Bad 👎🏻

```
function square(num) {
  if (typeof num === 'undefined') {
    return undefined;
  }
  return num ** 2;
}
```

💩 三角法则

就像鸟巢，鸟巢，鸟巢。

Good 👍🏻

```
function someFunction() {
  if (condition1) {
    if (condition2) {
      asyncFunction(params, (result) => {
        if (result) {
          for (;;) {
            if (condition3) {
            }
          }
        }
      })
    }
```

```
  }
}
```

Bad 👎

```
async function someFunction() {
  if (!condition1 || !condition2) {
    return;
  }

  const result = await asyncFunction(params);
  if (!result) {
    return;
  }

  for (;;) {
    if (condition3) {
    }
  }
}
```

💩 混合缩进

避免缩进，因为它们会使复杂的代码在编辑器中占用更多的空间。如果你不喜欢回避他们，那就和他们捣乱。

Good 👍

```
const fruits = ['apple',
  'orange', 'grape', 'pineapple'];
  const toppings = ['syrup', 'cream',
                    'jam',
                    'chocolate'];
const desserts = [];
fruits.forEach(fruit => {
toppings.forEach(topping => {
    desserts.push([
fruit,topping]);
    });})
```

Bad 👎

```
const fruits = ['apple', 'orange', 'grape', 'pineapple'];
const toppings = ['syrup', 'cream', 'jam', 'chocolate'];
const desserts = [];

fruits.forEach(fruit => {
```

```
  toppings.forEach(topping => {
    desserts.push([fruit, topping]);
  });
})
```

💩 不要锁住你的依赖项

以非受控方式更新每个新安装的依赖项。为什么坚持使用过去的版本，让我们使用最先进的库版本。

Good 👍🏻

```
$ ls -la
package.json
```

Bad 👎🏻

```
$ ls -la
package.json
package-lock.json
```

💩 函数长的比短的好

不要把程序逻辑分成可读的部分。如果IDE的搜索停止，而您无法找到所需的文件或函数，该怎么办?

- 一个文件中10000行代码是OK的。
- 一个函数体1000行代码是OK的。
- 处理许多服务(第三方和内部，也有一些工具、数据库手写ORM和jQuery滑块)在一个' service.js ' ?这是OK的。

💩 不要测试你的代码

这是重复的并且不需要的工作。

💩 避免代码风格统一

编写您想要的代码，特别是在一个团队中有多个开发人员的情况下。这是一个“自由”的原则。

💩 构建新项目不需要 README 文档

一开始我们就应该保持。

💩 保存不必要的代码

不要删除不用的代码，最多是注释掉。

以上都是💩 一样的代码。相信不少的初学者，或者是一些老鸟也会产生这样的问题。这里给大家提个醒，**切记不能这么写！！！**