代码风格和压缩

代码压缩

- 精简代码体积，减小网络传输时间，提高页面响应。
- 删除代码注释
- 删除无意义或者多余的空白（如空格，制表符，回车，换行）
- 删除可以省略的符号（如 css 最后一条规则后面的分号，js 块内最后的一条语句的分号）
- 缩短语句（如果 css 的简写，html 中 disabled='disabled' 改成 disabled , js 中缩短局部变量）
- 混淆: 降低代码的可读性，防止被追踪出程序逻辑。

代码风格

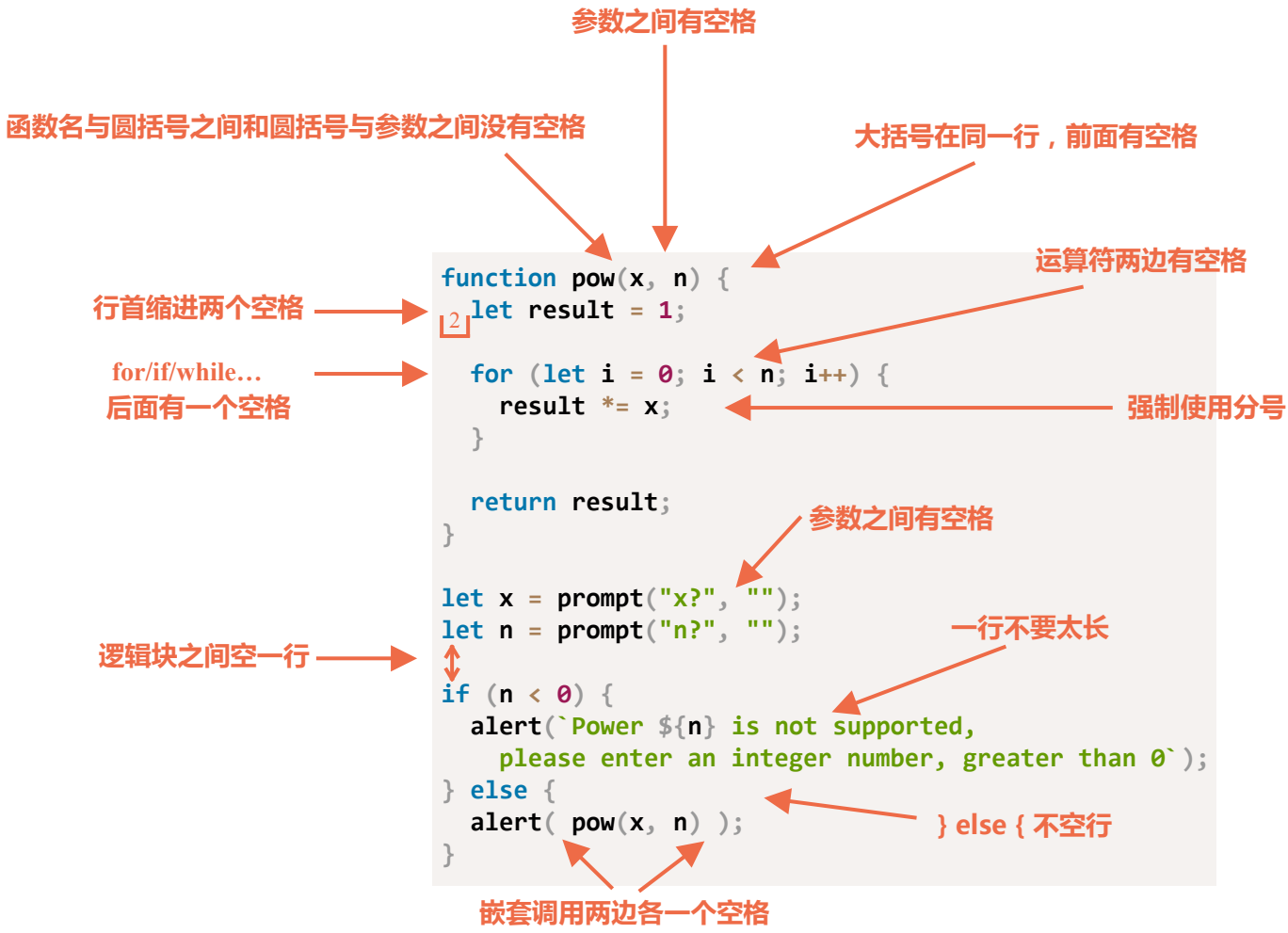
我们的代码必须尽可能的清晰和易读。

这实际是一种编程艺术 —— 以一种正确并且人类易读的方式编码来完成一个复杂的任务。

有一个帮助你（实现上面的目标）的事情就是良好的代码风格。

语法

一个含有规则的备忘录（更多细节如下）：



首先，我们需要明确一下，没有什么规则是“刻在石头上”的，每一条都是可选的而且可修改的：这些是编码规则，而不是宗教教条。

大括号

在大多数的 JavaScript 中，大括号（的开始部分）都是写在同一行而不是新换一行。这就是所谓的 "egyptian" 风格。（"egyptian" 风格又称 K&R 风格 —— 代码段括号的开始位于一行的末尾，而不是另外起一行的风格。）对了，在括号的开始部分前面还有一个空格。

例如：

```
if (condition) {
        // do this
        // ...and that
        // ...and that
}
```

总结：

- 对于很短的代码，一行是可以接受的：例如 `if (cond) return null`.
- 但是括号中的每个语句单独一行通常更好些。

缩进

有两种类型的缩进：

- **水平方向上的缩进: 2(4) 个空格。**
  一个水平缩进通常由 2 或 4 个空格或者 "Tab" 制表符构成。选择哪一个方式是一场古老的圣战。
- **垂直方向上的缩进：用于将逻辑块中的代码进行分割的空行。**
  即使是单个函数通常也被分割为数个逻辑块。在下面的例子中，初始化的变量、主要的循环结构和返回值都被垂直分割了。

  ```
  function pow(x, n) {
          var result = 1;
          //              <--
          for (var i = 0; i < n; i++) {
                  result *= x;
          }
          //              <--
          return result;
  }
  ```

  插入一个额外的空行有助于让代码更加地易读。连续超过 9 行都没有被垂直分割的代码是不应该出现的。

分号

每一个语句后面都应该有一个分号。即使它可能会被跳过。

嵌套的层级

你不应该嵌套太多的层级。

例如，下面的两个结构是相同的。

第一个：

```
function pow(x, n) {
        if (n < 0) {
                alert("Negative 'n' not supported");
        } else {
                var result = 1;

                for (var i = 0; i < n; i++) {
                        result *= x;
                }

                return result;
        }
}
```

还有这个：

```
function pow(x, n) {
        if (n < 0) {
                alert("Negative 'n' not supported");
                return;
        }

        var result = 1;

        for (var i = 0; i < n; i++) {
                result *= x;
        }

        return result;
}
```

但是第二个更加的可读，因为 `n < 0` 这个”边缘情况“已经提前被处理过，并且我们有一个 ”主“ 代码流，而不需要额外的嵌套。

函数在代码下面

如果你正在写几个”辅助类“的函数和一些使用它们的代码，有三种方式来放置它们。

1. 函数在调用它们的那些代码之上：

```
// *!*函数声明*/!*
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}

// 使用函数的代码
var elem = createElement();
setHandler(elem);
walkAround();
```

2. 先写代码，再写函数

```
var elem = createElement();
setHandler(elem);
walkAround();

function createElement() {
...
}
function setHandler(elem) {
...
}
function walkAround() {
...
}
```

大多数时候，第二种方式更好。

这是因为当在阅读代码时，我们首先想要知道的是“它做了什么”。如果代码先行，它就会提供这些信息。或许我们一点也不需要阅读这些函数，尤其是他们的名字足够表示出他们做了什么的时候。

风格指南

风格指南包含了“如果编写代码”的通用规则：使用什么引号、用多少空格来缩进、哪里放置换行等等很多的小细节。

总的来说，当团队中的所有成员都使用同样的风格指南时，代码看起来是统一的。无论团队中谁写的，都是一样的风格。

当然，一个团队可能会考虑一个他们自己的风格指南。但是现在，他们没必要这样做。现在有很多已经尝试过并制作好的风格指南，可以很容易采用。

例如：

- Google JavaScript 风格指南
- Airbnb JavaScript 风格指南
- Idiomatic.JS
- StandardJS
- (还有很多)

自动检测器

有很多工具可以自动检查你的代码风格。他们叫做 "linters"。

它们有一个很棒的地方是风格检测也会发现一些 bug（问题），诸如变量名或者函数书写错误。

最出名的工具有：

- JSLint -- 第一批 linters 之一。
- JSHint -- 比 JSLint 多了更多设置。
- ESLint -- 可能是最新的一个。

总结

本章的所有语法规则和样式指南旨在提高可读性，因此所有的内容都是值得商榷的。

当我们思考“如何写地更好”的时候，唯一的标准是“什么会让代码更加可读和容易理解，什么会帮助我们避免错误”。这是当选择一种风格或讨论哪一种更好的时候要牢记的主要原则。