
如何写好一篇技术型文档

周智 2022-1-20



参加工作时间久一点的工程师应该有这样一个体会：自己平时代码写得再多再好，可一旦要用文档去描述或者表达某一个事情或者问题时，都感觉非常困难，无从下手，不知道自己该写什么不该写什么；或者费了九牛二虎之力写出来的东西没法满足要求，需要再三去修改调整。这其中的主要原因我归纳有两点：

1. 思维方式固化。大部分人平时代码写得太多，文字类型的表述又写得太少。而代码和文字明显是两种不同的思维方式，在代码里陷得太深，不容易跳出来；
2. 本身文字表达能力有限。这个跟写代码一样，有人代码质量高、bug 少；有人水平低、bug 自然就多。

以上两点其实都可以通过平时多练、多写、多梳理的方式去弥补，比如周期性的博客总结和记录。但是，如果你能刻意系统性地去补充一些关于“技术型写作”的理论知识，一定能够事半功倍。这就像我们刚学编程时，一顿学、一顿模仿，但是总感觉缺了点什么，自己再努力发现深度还是不够。这时候，我们需要做的是看一本高质量的经典书籍，书籍能帮我们梳理知识点、总结各种碰到过的问题，从理论上解答我们心中各种疑惑，将之前的野路子“正规化”。

下面是我根据平时的一些积累，将技术型写作的理论知识归纳成 10 个要点（点击跳转）：

-
1. 搞清楚主谓宾
 2. 不滥用代词、过渡词和标点符号
 - 2.1 不滥用代词和过渡词
 - 2.2 不滥用标点符号
 3. 多用强势动词，少用形容词和副词
 - 3.1 强势动词和主动语句
 - 3.2 少用形容词和副词
 4. 正确使用术语
 5. 正确使用段落
 - 5.1 单一职责
-

-
- 5.2 好的开头语
 - 5.3 控制段落长度
 - 6. 适当使用列表和表格
 - 6.1 使用列表
 - 6.2 使用表格
 - 7. 一图胜千言
 - 7.1 可抽象也可具体
 - 7.2 突出图中重点
 - 7.3 有准确的图标题
 - 8. 统一样式和风格
 - 9. 把握好整体文档结构
 - 10. 明确文档的目标群体
-

1 搞清楚主谓宾

文档主要由段落组成，段落由句子组成，而大部分句子又由“主谓宾”组成（可能有些场合省略了，但是读者可以通过上下文轻松 **get** 到）。主谓宾是主干骨架，其他内容可以看作是句子的修饰，主干骨架是决定句子是否垮掉的主要原因。现在很多人可能已经忘记了句子的基本构成，毕竟以汉语为母语的人，大概率是不太会关心这些“细节”，就像说英语的国家可能不太关心 **am is are** 一样，你说哪个人家都理解。但是，文档中的一句话读起来是否别扭，大多数时候是由句子构成决定的。在不考虑文档上下文的情况下，如果一个句子能包含正确的主语、谓语和宾语（可选），那么它读起来至少是很顺口的。下面举一个明显搞不清主谓宾的例子：



传统图像处理算法，通过计算烟火颜色特征，极易受烟火周围环境相近颜色干扰而造成误检。

尽管你能读懂作者想要表达的意思，但是这句话读起来还是太别扭。“传统图像处理算法”应该算是主语，后面的“通过...”这句不完整，“极易受...干扰”这句还可以，“...造成误检”算是谓语宾语，但是这里用错了动词，为什么是“算法造成误检”，难道不是“周围环境相近颜色干扰造成误检”吗？

这句话的主干内容是：算法极易受...影响而...。正确的表述应该类似下面这样：



因为传统图像处理算法通过计算烟火颜色特征去识别烟火，所以它极易受烟火周围环境相近颜色干扰而出现误检。

我们用过渡词（因为...所以...）将原来的句子拆成了前后两个部分，前面部分的主语是“传统图像处理算法”，谓宾是“识别烟火”；后半部分的主语是“它”，谓宾是“出现误检”。经过调整后，前后两个部分的主语是同一个：传统图像处理算法。下面再直观看一下修改之后的句子主干骨架：

因为传统图像处理算法通过计算烟火颜色特征去识别烟火，
所以它极易受烟火周围环境相近颜色干扰而出现误检。

如果你觉得用“因为...所以...”不太好，那么可以再换一种表述：



传统图像处理算法通过计算烟火颜色特征去识别烟火，烟火周围环境相近颜色的干扰极易造成误检。

第一句还是跟之前一样，主语是“传统图像处理算法”，第二句主语变成了“干扰”，谓宾是“造成误检”。下面我们直观地看一下修改之后的句子主干骨架：

传统图像处理算法通过计算烟火颜色特征去识别烟火，
烟火周围环境相近颜色的干扰极易造成误检。

最后再举一个错误的例子：



由于误报率与漏报率很高，因此不管是否有真实事件发生都会去留意，也会有规定的日程定点巡查视频任务。

上面这个句子的作者完全没搞懂谁是主语，谁是谓语。感兴趣的童鞋可以试着修改一下，改成你认为正确的表述。

2 不滥用代词、过渡词和标点符号

2.1 滥用代词和过渡词

中文文档中的代词主要有：你我他她它、其、前者、后者、这样、那样、如此等等，过渡词主要有：因为/所以、不但/而且、首先/然后等等。下面这张表格列举了一些常见的代词和过渡词及其常用场合：

表 2-1 代词和过渡词举例

序号	类型	名称	常用场合举例
1	代词	其	C 语言中引入了“指针”的概念， 其 作用是为了能够提升内存访问速度。
2	代词	后者	C 语言发明于 1970 年代，C++ 语言发明于 1980 年代， 后者 主要引入了面向对象思想。
3	代词	此	指针能够提升程序访问内存的速度，但 此 特点仍存在一些缺陷。
4	代词	它	C 语言的一大特性是指针，这就像 C++ 语言和 它 的面向对象思想一样。
5	过渡词	因为/所以	因为 神经网络可以自动提取数据特征， 所以 基于神经网络的深度学习技术中不再有传统意义上的“特征工程”这一概念。
6	过渡词	首先/然后	首先 我们要保证有足够多的训练数据， 然后 我们再选择一个适合该问题的神经网络模型。

代词和过渡词就像标点符号一样，容易被滥用。代词滥用主要体现在作者在使用它们的时候并没有搞清楚它们代表的究竟是谁，是前一句的主语、还是前一句的宾语或者干脆是前一整句话？过渡词滥用主要体现在作者在使用它们的时候并没有搞清楚前后两句话的逻辑关系，是递进还是转折或者是因果？（过渡词滥用频率要低很多，毕竟搞清楚前后句子逻辑的难度要小）接下来举几个滥用代词和过渡词的例子：



C++语言发明于 1980 年代，它支持“指针”和“面向对象 (Object-Oriented)”两个特性，其价值在计算机编程语言历史上数一数二。

上面这个句子中出现了两个代词“它”和“其”，抛开句子内容本身对错不论，第二个代词指向的对象其实并不明确，“其”指的是“指针”、“面向对象”还是“C++语言”？或者是指“C++语言同时支持...两个特性”这个陈述？像这种有歧义の場合，我们应该少用代词，尽量用具体的主语去代替：



C++语言发明于 1980 年代，它支持“指针”和“面向对象 (Object-Oriented)”两个特性，C++ 的价值在计算机编程语言历史上数一数二。

如果你一定要用代词，那么调整一下可能更好：



C++语言发明于 1980 年代，它同时支持“指针”和“面向对象 (Object-Oriented)”两个特性，这个价值在计算机编程语言历史上数一数二。

再读一读，你是不是没有感觉到歧义了？我们在“支持”前面增加了一个“同时”，然后将代词换成了“这个”，现在这个代词指的是“C++语言同时支持...两个特性”这个陈述，修改后整个句子的意思更明确。

我们再来看另外一个滥用代词的例子：



该模块主要负责对视频进行解码，输出单张 YUV 格式的图片，并对输出的图片进行压缩和裁剪，前者基于 Resize 方法来完成，后者基于 Crop() 方法完成。

对于大部分人来讲，上面这段没什么问题。代词“前者”指的是压缩、“后者”指的是裁剪，原因很简单，因为单词 Resize 对应的是压缩、单词 Crop 对应的是裁剪。但是这段话如果拿给没有任何知识背景的人去读（大概率可能是找不到这种人），恐怕会存在歧义，主要原因是代词前面提到了很多东西，“前者”和“后者”指向不明确，到底是指“解码”、“输出单张图片”还是后面的“压缩”和“裁剪”？下面这样调整后，整段话的意思更加明确：



该模块主要负责对视频进行解码，输出单张 YUV 格式的图片，并对输出的图片进行压缩和裁剪，

压缩基于 Resize 方法来完成，裁剪基于 Crop() 方法完成。

我们去掉了代词，直接用具体的主语来代替，句子意思非常明确。如果你一定要使用代词，那么也可以这样调整：



该模块主要负责对视频进行解码，输出单张 YUV 格式的图片。同时，它还对输出的图片进行压缩和裁剪，前者基于 Resize() 方法完成，后者基于 Crop() 方法完成。

上面这段话还是使用了代词“前者”/“后者”，但是我们修改了标点符号，并且增加了一个过渡词“同时...”，这样做的目的是让读者知道虽然整段话说的是同一个东西，但是前后的句子已经分开了，为我们后面使用代词做好准备。

好的，现在我们来总结一下在技术型文档编写过程中使用代词时的一些有价值经验：

- ✓ 代词可以指它前面出现过的名词、短语甚至整个句子，但是一定是前面出现过的；
- ✓ 代词的位置和它要指向的目标最好不要隔得太远，1~3 句话之内，超过就不要用了；
- ✓ 代词的作用是减少小范围内某些词汇或句子重复出现的频率，要用到恰到好处；
- ✓ 代词前面出现的混淆目标如果太多，一定要重新调整句子，确保代词指向无歧义。

2.2 滥用标点符号

接下来我们再看另一个，标点符号的滥用要普遍很多，其主要原因是：标点符号的使用并没有非常明确的对错之分。至少对大部分人而言，使用句号还是逗号其实并没有什么严格的评判标准，只要不出现“一逗到底”的极端情况，其余大概率都 OK。下面这张表格是我根据以往经验，总结出来的应用于技术型写作时中文标点符号使用规则：

表 2-2 常用标点符号

序号	符号	写法	使用场合
1	逗号	,	前后两句话关联性比较大，阅读时停顿时间短。

序号	符号	写法	使用场合
2	句号	。	前后两句话关联性比较小，阅读时停顿时间稍长。
3	分号	；	前后两句话地位相对平等，句子的内容和格式基本保持一致。比如列表中，如果每项是一个句子或者短语，那么第 1 至第 N-1 项结尾使用分号，第 N 项结尾使用句号。
4	冒号	：	技术型文档中，冒号一般用在需要引入重要内容的场合。比如当你需要插入一张表格或者一张图片时，需要提前做一个提醒（下表列举了常见的代词和过渡词：），提醒结束时补充一个冒号。
5	括号	（） 【】	（）一般用于解释性的场合，负责对名词或者句子的补充解释。 【】用得比较少，我一般用于需要增加醒目标记的名词或短语中。
6	顿号	、	一般可以用在枚举名词或者短语的场合。
7	问号	？	不用多解释。
8	引号	“” “，”	一般用于标记特殊名词、专用名词、短语，或需要重点突出的名词或短语。
9	分隔号	/	一般用于成对出现的名词（举例：因为/所以、首先/然后等等都是过渡词），或者根据文档上下文来判断地位差不多的相近词（举例：算法的好坏直接影响最终报表中误报/误报率那一栏）。
10	破折号	——	用得不多。
11	省略号	...	不用多解释。
12	感叹号	！	技术型文档不是写小说，用得不多。
13	书名号	《》 〈〉	不用多解释。

上面这张表格基本涵盖了常用的中文标点符号，其中有一小部分在技术型文档中不太常见，比如感叹号、破折号，这些符号多多少少带有某种感情色彩，不太适合用于技术型文档编写。前面已经简单概括了一下各个符号的使用场合，下面挑几个容易出错的再一一详细说明：



C++语言发明于 1980 年代，它衍生自 C 语言，主要引入了“面向对象（Object-Oriented）”思想，面向对象思想强调对数据的封装和对功能的复用，此特性有利于开发者对代码的维护和扩展，目前，大部分计算机编程语言已经支持了面向对象特性。

上面这段话属于典型的“一逗到底”的例子。作者从 C++ 语言说到了面向对象思想，最后总结大部分计算机编程语言都支持面向对象。我们如果将整段话拆开来看，其实它想表述的是 3 个内容，每个内容之间最好使用句号，停顿时间稍长一些。我们调整之后的效果是：



C++语言发明于 1980 年代，它衍生自 C 语言，主要引入了“面向对象（Object-Oriented）”思想。面向对象思想强调对数据的封装和对功能的复用，此特性有利于开发者对代码的维护和扩展。目前，大部分计算机编程语言已经支持了面向对象特性。

接下来我们再看看分号的使用。根据我个人经验，分号常用在列表场合，下面举一个例子说明：



下面是“将大象装入冰箱”的具体步骤：

1. 打开冰箱门；
2. 将大象装进冰箱；
3. 关上冰箱门。

上面是一个有序列表，列表中的各项内容是一个短语。当列表中各项内容是短语或者句子的时候，除最后一项之外其余项目结尾一般都使用分号（注意，同一个列表中各项的格式最好都保持一致，要么都是短语，要么都是单个的名词，这个后面专门讲列表的时候会提到）。如果列表中各项内容只是一个名词时，那么结尾就可以不用标点符号：



下面是“可以被装进冰箱”的动物：

- ✓ 狗子
- ✓ 大象
- ✓ 猴子
- ✓ 鹦鹉

上面是一个无序列表，列表中的各项内容是一个名词，这时候名词结尾处不需要添加任何标点符号。

我们最后再来看一下小括号的使用场合。在技术型文档中，小括号主要用于对前面的名词、短语或者句子进行补充说明，比如当文档中出现缩写词汇时，我们会在它的后面增加一个小括号，在括号里面注明该缩写词汇的全称。下面举一个使用小括号对缩写词汇解释说明的例子：



API（Application Program Interface）是系统对外提供的访问接口，使用者可以按照 API 文档中的接口定义去访问系统中的数据，并与它做一些交互。

上面这段话主要讲 API 是什么、可以干什么。它是 Application Program Interface 三个单词的简称，为了让读者更清楚该术语的定义，作者可以选择在第一个“API”出现的位置增加一个小括号，并将术语全称补充进来，之后的整个文档无需再重复该操作（后面会单独提到术语全称和简称的运用规则）。

除了能对缩写词汇进行解释说明之外，小括号还可以用于对前面整个句子进行补充说明，再看下面这个例子：



它是 Application Program Interface 三个单词的简称，为了让读者更清楚该术语的定义，作者可以选择在第一个“API”出现的位置增加一个小括号，并将术语全称补充进来，之后的整个文档无需再重复该操作（后面会单独提到术语全称和简称的运用规则）。

上面这段话其实前面已经出现过，最后小括号里面的内容主要是为了对它前面一句话进行补充。如果补充性说明内容太长，比如要好几句话才能起到补充的作用，那么这个时候我们就不应该再使用小括号了，可以考虑调整句子结构，然后将补充性的内容当作段落主体的一部分。

关于代词、过渡词以及标点符号滥用的内容就讲到这里，其中有一些内容是我个人的写作喜好，其实并没有非常明确的对错之分，比如前面讲到列表中分号的使用，很多人这时候可能选择使用句号。大家可以根据自己的判断去处理这种模棱两可的场景，当然一些比较确定的规则，比如当列表项只有名词的时候，列表项结尾不要使用任何标点符号，这一点还是比较确定的。

3 多用强势动词，少用形容词和副词

3.1 强势动词和主动语句

很多人可能第一次听到“强势动词”这个说法，陌生还难以理解。如果将它翻译成英文，对应的单词应该是“Strong Verbs”，意思是强有力的动词，你可以理解为：听起来动作幅度大、冲击力强的那一类动词。打个比方，假如“走”是弱势动词，那么“跳”就是强势动词；假如拿刀“切”是弱势动词，那么拿刀“砍”就是强势动词。下面这张表格列举了一些强势/

弱势动词的例子：

表 3-1 强势/弱势动词对比

序号	弱势动词	(可考虑) 强势动词
1	走过去	跳过去
2	切肉	砍肉
3	出现异常	抛出异常
4	程序退出	程序崩溃
5	内存增长	内存泄漏
6	找不到日志文件	日志文件丢失
7	客户提出质疑	客户投诉
8	任务未完成	任务延期
9	角色权限是由管理员设置的	管理员控制角色权限
10	系统无法正常使用 API 返回的结果	系统无法正常解析 API 返回的结果

上面列出了 10 对强势/弱势动词，我们观察可以发现：弱势动词一般无法正确表达问题/事情的真实情况。在技术型文档编写过程中，虽然我们不能借助词汇使用、句子构成以及标点符号等手段去传递感情倾向，但是也不能掩盖真实准确的内容表达。

在提到强势动词时，我们还要注意“主动语句”和“被动语句”的区别。在技术型文档编写过程中，应该尽量少使用被动语句。下面这张表格列举了一些主动/被动语句的例子：

表 3-2 主动/被动语句对比

序号	被动语句	(可考虑) 主动语句
1	角色权限是由管理员控制的	管理员控制角色权限
2	API 结果无法被系统正常解析	系统无法正常解析 API 结果
3	图像特征是通过 CNN 逐步降维的方式提取的	CNN 通过逐步降维的方式提取图像特征
4	这种检测效果无法被客户接受	客户无法接受这种检测效果
5	经过研发排查发现，这个现象是正常的 (*)	经过研发排查发现，这个属于正常现象

上面表中第 5 项（带*号）严格来讲不算被动语句，但是在技术型写作过程中，我们应该避免使用“...是...的。”这种句式，该句式太过口语化。尽量少用被动语句的原因有以下三个：

- ✓ 读起来麻烦。读者读到被动语句时，需要先在脑子里将其转换一下再去理解；

- ✓ 难以理解。读者有时候很难分清被动语句中的真实主语（甚至可能省略了主语）；
- ✓ 字数多。被动语句一般更长、字数更多。

那么被动语句是不是完全不让用了呢？当然不是。仔细的读者可能已经观察到了前面在举例的时候我们有这样一段话：

C++语言**发明于** 1980 年代，它支持“指针”和“面向对象 (Object-Oriented)”两个特性，C++ 的价值在计算机编程语言历史上数一数二。

上面第一句中的“...于”其实就是被动语句，像“C++语言发明于...”、“该文档编辑于...”这些都算被动语句，由于宾语（这里是 C++ 语言）更重要，所以默认省略了真实主语（某某发明 C++ 语言，可是某某在这里不太重要）。这类句子结构有一个特点就是：宾语比真实主语重要，所以放到句子的开头位置。

3.2 少用形容词和副词

技术型文档讲究的是一个“准”字，它不像小说、散文之类的文学作品带有很强的感情色彩，也不同于网络博客可以掺杂一些非正式词汇，更不能跟 Marketing Speech（营销话语）一样常常夸大其词。为了做好前面说的“准”，技术型文档应该尽量少用形容词和副词，因为这些词语大部分都属于“主观”表达。下面举几个使用形容词和副词的例子：



为了保证系统运行更高效，他们尝试尽可能压缩图片尺寸，事实证明这个尝试非常成功。这样的工作看似简单，却蕴含着高技术含量。

上面这段话使用了好几个副词和形容词，比如“尽可能”、“非常”、“高”。如果是技术型文档，这段话建议调整为：



为了提高系统运行效率，他们将图片尺寸压缩到原来的 1/3，系统响应速度提升 2 倍。

我们用具体的数值替换了原来的形容词和副词，并且直接删掉了最后一句话，最后一句话在技术型文档中起不到任何作用。下面这张表格列举了部分形容词和副词使用不恰当的场所：

表 3-3 形容词/副词使用不恰当举例

序号	形容词/副词	(可考虑) 调整为
1	经过优化, 接口响应速度提升明显	经过优化, 接口响应速度提升 2 倍
2	很多人反应现场误报很多	数据统计发现, 现场误报率为 11%
3	大部分客户投诉说系统很不好用	最近一个月有超过 50 个客户投诉说系统不好用
4	升级依赖库后, 该函数运行很快	将依赖库升级到 2.3.1 版本后, 该函数执行时间缩短到 100ms 以内
5	研发同事很辛苦, 每天加班很晚	研发同事很辛苦, 每天 23:00 之后才下班

最后, 我们来总结一下:

- ✓ 优先使用方便读者阅读理解的动词和句式 (强势动词和主动语句);
- ✓ 尽量少用形容词和副词, 用具体数值代替、或者调整句子表述。

4 正确使用术语

这里提到的术语分两种: 一种是计算机领域通用的专业术语, 像 SDK、面向对象、TCP/IP、微服务等等这些名词, 它们基本已经被大众接受和理解, 我们在编写文档的时候不能随意再重新去命名、调整或者改变拼写 (将 “TCP/IP” 写成 “Tcp/ip”); 另外一种是当前文档需要定义的术语, 这种术语只有在当前文档上下文中才有效。我们在编写技术型文档时, 通过自己的判断, 如果认为文档读者缺乏对相关术语 (不管是前面哪一种) 的理解, 我们都应该在文档靠前位置给出对术语的解释说明, 也就是我们平时文档中常见的 “名词解释”。

表 4-1 名词解释举例 (*为自定义术语)

序号	名词	说明
1	SDK	Software Development Kit, 软件开发包, 开发者基于该工具包开发更丰富的高层应用。
2	内存泄漏	通过 new/malloc 等方法申请的内存存在使用完后未被及时释放, 程序运行内存占用越来越高。
3	面向对象	强调对数据和功能的封装, 提升代码的可复用性、可扩展性以及灵活性。

序号	名词	说明
4	FVM (*)	Front Video Manager, 前端视频管理服务, 负责视频接入、分发等业务。
5	视频大数据标签服务 (*)	对视频进行结构化处理, 生成结构化标签, 并对外提供标签检索等功能。

有些文档可能篇幅比较短, 并不是传统意义上的需求设计类文档, 比如对某个线上问题分析的结果汇报、对某个模型检测效果的验证报告、或者研发阶段性的工作总结。这些文档由于本身内容就不多, 大部分可能直接进入主题, 这时候如果还要在文档中专门增加一块名词解释的版块 (并且总共也就一两个术语), 就显得比较突兀。

另外一种对术语进行解释说明的方式是用我们前面提到的小括号, 我们可以在术语后面增加一个小括号, 然后在括号里添加补充说明。这种方式很便捷, 但是只适合简单的场景, 比如在小括号里面补充术语的全称或者简称, 或者只做简单的解释说明。如果对一个术语的解释内容很长, 就不太适合用这个方法, 下面举一个错误的例子:



当视频离线时, FVM (Front Video Manager, 前端视频管理服务, 负责视频接入、分发等业务。) 会产生一条告警记录, 并存入节点数据库。

上面这个术语解释内容太长, 不太适合使用小括号的方式, 这种情况要么在文档正文中专门对 FVM 进行解释, 要么在小括号中只给出 FVM 的英文全称即可:



当视频离线时, FVM (Front Video Manager) 会产生一条告警记录, 并存入节点数据库。

使用小括号去做术语解释还需要注意一点的是: 只需要在术语第一次出现的时候做一次解释即可, 不需要重复多次。下面举一个重复的错误例子:



当视频离线时, FVM (Front Video Manager) 会产生一条告警记录, 并存入节点数据库。之后节点数据库会将该条告警记录同步到平台数据库, 平台 FVM (Front Video Manager) 检测到有新的告警记录时, 会通过消息中间件通知业务系统, 业务系统随后将告警信息以短信 (或钉钉) 的方式通知到用户。

上面对术语 FVM 的解释重复了两次, 这种做法是错误的, 第二次我们可以直接去掉。

有些术语存在全称和简称, 我们熟悉的 SDK 全称是 “Software Development Kit”, 但是现在

基本没有人再去使用它的全称。像这种简称已经被大众熟知的术语，我们就不能再标新立异的去用它的全称。另外一些在文档中自定义的术语，文档作者为了便于阅读可能也会提供一个简写的版本，在这种情况下，文档前后应该保持一致，即：要么整篇文档都用全称，要么都用简称，尽量做到一致。下面举一个全称简称使用不一致的例子：



IVA (Intelligent Video Analytics, 智能视频分析) 服务主要负责视频解码、模型推理、目标跟踪以及目标行为分析，该服务是整个系统中最复杂的一个模块。智能视频分析服务由张三团队开发完成，一共耗时 6 个月，人力成本开销 120 万。

上面这段话中，前半部分作者使用“IVA”简称（小括号中做了全称说明），但是在后面一句话中作者又使用了全称“智能视频分析”，这种做法没有遵循统一原则。不仅同一段落应该保持统一，整篇文档也应该做到统一，术语在文档中第一次出现时是简称，那么整篇文档都应该用简称，反之亦然。

最后我们来总结一下，在技术型文档中使用术语时需要注意的一些事项：

- ✓ 文档读者不熟悉的术语（包括通用术语和文档自定义术语）都应该有解释说明；
- ✓ 小括号只适合简短的术语解释场合，括号里的内容不能太长（一两句短语之内）；
- ✓ 任何方式的术语解释只需要有一次即可（术语第一次出现时），不要解释多次；
- ✓ 术语的全称和简称要保持使用一致，要么整篇文档都用全称、要么都用简称；
- ✓ 对于计算机领域的通用专业术语，需要沿用主流用法，不要随意再去调整。

5 正确使用段落

5.1 单一职责

与面向对象编程中“类的单一职责原则”一样，文档中的句子（特指以句号结尾的一句话）、段落也应该遵循“单一职责原则”。前面讲标点符号的时候已经提到过，同一段话中前后关联性不大的两句话之间用句号，这样可以保证每句话想要表达的是相对独立的内容。段落也

一样，一个段落只陈述一个主题，可以保证段落的句子不会太多、内容不会太长，便于读者阅读和理解。下面举一个段落使用错误的例子：



Excel 提供一个组织数据的高效方法。我们可以将 Excel 想象成一个有行和列的二维表格，每一行代表一个独立的实体，每一列代表该实体的不同属性。Excel 还具备数学功能，比如计算平均值和方差等数学操作。如果你想使用 Excel 来记录图书信息，那么每一行代表不同的书本，每一列代表书本的属性，比如书的名称、价格以及出版社等等信息。

上面这段话的第一句已经明确了段落主题：Excel 能高效地组织数据。可是，这段话中间却穿插了一个不相干的句子，说 Excel 具备数学功能，能够做一些数学操作，这句话显然跟本段主题不一致，我们需要将其去掉：



Excel 提供一个组织数据的高效方法。我们可以将 Excel 想象成一个有行和列的二维表格，每一行代表一个独立的实体，每一列代表该实体的不同属性。~~Excel 还具备数学功能，比如计算平均值和方差等数学操作。~~如果你想使用 Excel 来记录图书信息，那么每一行代表不同的书本，每一列代表书本的属性，比如书的名称、价格以及出版社等等信息。

5.2 好的开头语

除了要保证段落的“单一职责”之外，我们还需要给每个段落一句“好的”开头语。那么什么是好的开头语呢？好的开头语要能让读者读完之后就能猜到文档作者在本段中想要陈述的主题，其实就是概括性的句子。还是以上面那段话为例子，它的第一句话“Excel 提供一个组织数据的高效方法”其实就是很好的开头语，它提示本段内容主要讲 Excel 如何高效地组织数据。如果我们将上面那段话的开头调整一下，那么效果明显就差了很多：



Excel 由许许多多的单元格组成，每个单元格可以包含不同的内容。我们可以将 Excel 想象成一个有行和列的二维表格，每一行代表一个独立的实体，每一列代表该实体的不同属性。如果你想使用 Excel 来记录图书信息，那么每一行代表不同的书本，每一列代表书本的属性，比如书的名称、价格以及出版社等等信息。

读者读完上面第一句话后，可能还是很懵，需要读完整段话才能明白文档作者在本段中想要

表达的意思。段落的开头语可以通过提炼段落内容得到，我们可以在段落写完之后回过头提炼一句话作为本段的开头语，下面这段话描述代码中循环语句的作用：



目前几乎所有的计算机编程语言都支持循环语句，例如，我们可以编写代码来判断一个用户命令行输入是否等于“quit”（退出命令），如果需要判断 100 万次，那就创建一个循环，让判断逻辑代码运行 100 万次。

上面的这段话本身没什么问题，主要介绍循环语句的功能和应用场合。但是如果我们提炼一下，在段落开头增加一个更好的开头语，效果可能会提升很多：



循环语句会多次运行同一个代码块，直到不再满足循环条件为止。目前几乎所有的计算机编程语言都支持循环语句，例如，我们可以编写代码来判断一个用户命令行输入是否等于“quit”（退出命令），如果需要判断 100 万次，那就创建一个循环，让判断逻辑代码运行 100 万次。

上面开头第一句话就说清楚了循环结构的特点，读者读完第一句话基本就知道整段内容要讲什么。一个好的开头语能够节省读者的时间，因为并不是每个读者都有兴趣去阅读整段的内容，开头语可以给读者“是否继续读下去”一个参考。

5.3 控制段落长度

控制段落长度并没有一个明确的标准，它只是一个非常主观的说法。如果文档中某个段落内容太长（比如那种一段话就占半页 Word），作者自己应该反复阅读几次再对段落做一些精简，这样既可以节省读者的时间，大概率也能提升意思表达的准确性。同样，也不太建议文档频繁出现小段落，比如整段内容只有一两句话那种，这个时候可以考虑段落合并或者稍微扩充一下内容。

最后我们来总结一下，在技术型文档中如何正确使用段落：

- ✓ 一个段落只负责讲一个内容，两个不同的主题应该拆分成两个段落去陈述；
- ✓ 尽量为每个段落增加一个“好的”开头语，能够清晰表达（或暗示）本段的主题；
- ✓ 要控制好段落内容长短，“不长不短”根据自己经验（比如不超 5~7 个句子）。

6 适当使用列表和表格

文字相对来讲其实是一种效率比较低的表达方式。如果你想让人快速地去理解你要表达的意思，图片应该是最好的一种方式，但是图片有一个缺点就是：有时候它只能从宏观上去表达，无法体现其中细节。当我们想要尽可能直观地去陈述内容，又想尽可能多的包含细节时，我们可以考虑使用列表或者表格。有些读者非常抵触大段大段的文字（尤其在技术型文档中），一种改进方法是前面提到的“控制段落长度”，尽量让段落内容精简、单一；再一个就是看看段落内容是否能以列表或者表格的方式去呈现，这种方式可以给人“严谨、清晰”的感觉。

6.1 使用列表

列表简单来讲就是将你原来用段落方式呈现的内容改用项目（Item）的方式去呈现，一般它主要用于枚举、过程描述或者要点归纳等场合。列表中的各项可以是名词、短语，甚至是句子，各项目之间有严格顺序要求的列表叫“有序列表”，相反并没有严格顺序要求的列表叫“无序列表”。下面是以段落的方式陈述小张今天所做的事情：

白天在公司上班期间，小张一共修复了 7 个 bug，做了 3 个代码合并（评审），并和项目经理讨论了前天的新需求。晚上回到家后，小张先做饭，然后给儿子洗澡，23:30 上床睡觉。

上面这段话本身没什么问题，用了合理的标点符号和过渡词，读起来清晰明了。但是，如果在技术型文档编写中，能将这段话改用列表的方式呈现，起到的效果会更好：



小张白天在公司：

- ✓ 修复了 7 个 bug；
- ✓ 做了 3 个代码合并（评审）；
- ✓ 和项目经理讨论前天的新需求。

晚上回到家后：

1. 做晚饭；
2. 给儿子洗澡；
3. 23:30 上床睡觉。

我们将原来的一段话拆成了两个列表，并在每个列表前面做了一个“引入说明”（以冒号结

束)，介绍了接下来列表的背景上下文。第一个列表是无序列表，因为原文并没有突出强调小张白天在公司每项工作之间的前后关系（无顺序要求），只是一个归纳统计；第二个列表是一个有序列表，原文很明显强调了小张晚上回家之后做事的先后顺序（最后一项还给出了具体时间）。在技术型文档中，合理地运用列表这种方式去呈现内容可以给人一种“逻辑严谨、思路清晰”的感觉，让读者更相信你讲的内容。

在使用列表时，我们应该确保列表中各项内容结构一致，即：要么都是名词，要么都是短语，要么都是句子。这个原则既能保证你使用列表的初衷（逻辑严谨、思路清晰），也能让读者读起来更舒服。下面是一个错误使用列表的示范：

影响系统检测准确性的因素有：

- ✓ 模型；
- ✓ 产品开通过程中，工程师对算法参数校准程度；
- ✓ 应用现场是否有灯光照明。

上面列表一共包含 3 项，每项的内容结构各不相同，第一项是一个名词，第二项是一个句子，第三项是一个短语。我们将结构统一后，可以调整为下面这样：



影响系统检测准确性的因素有：

- ✓ 模型的复杂性；
- ✓ 部署时对算法参数校准的程度；
- ✓ 应用现场是否有灯光照明。

上面是将列表中各项内容修改为短语，我们还可以换另外一种方式：



影响系统检测准确性的因素有：

- ✓ 模型类型
- ✓ 校准程度
- ✓ 环境亮度

上面是将列表中各项内容修改为名词，由于是名词，每项结尾处不使用任何标点符号（参见前面专门讲标点符号的章节）。下面是对列表运用的总结：

- ✓ 列表一般用于枚举、过程描述、要点归纳等场合；
- ✓ 需要强调顺序的时候应该使用有序列表，其余视情况而定；
- ✓ 列表中各项内容结构应保持一致，都是名词、短语或者句子；
- ✓ 每个列表前面尽量添加一个明确的“引入说明”，以冒号结束。

6.2 使用表格

表格其实跟面向对象有一定联系，大部分时候表格中的一行相当于一个对象，表格中的列相当于对象的属性（字段），表格和面向对象组织数据的方式本质上是一致的。技术型文档中表格一般用来组织与数字有关的内容，当然也有例外，就像前面章节中用到的表格，纯粹是为了组织文本内容。

下面是在技术型文档中，使用表格时可以参考的一些经验：

- ✓ 组织数字相关内容时，一定要用表格（大部分人可能已经有这个意识）；
- ✓ 组织结构化类型的文本内容时，尽量用表格；
- ✓ 每个表格都应该配一个表格标题，简要说明表格内容；
- ✓ 文档中的表格应具备一致的样式和风格，比如标题字体、背景填充等。

在技术型文档中使用表格组织文本内容时，需要控制每个单元格的文本长度。一般情况下建议单元格中只使用短语，如果必须要用段落，也应该控制段落中句子数量（一般建议不超过2~3句）。下面是错误使用表格来组织文本内容的示范：

表 6-1 三种编程语言介绍

序号	语言	介绍
1	C	C 语言由贝尔实验室发明于 1969 至 1973 年，是一种编译型计算机编程语言。它运行速度快、效率高、使用灵活，常被用于计算机底层驱动以及各种语言编译器的开发。C 语言是一种面向过程的编程语言，同时它的语法相对来讲较复杂，新人入门门槛比较高。
2	C++	C++语言发明于 1979 年，是一种编译型计算机编程语言。它衍生自 C 语言，

序号	语言	介绍
		继承了 C 语言的一些特性，比如使用指针直接访问内存，同时它也支持面向对象编程，提升了代码的可复用性、可扩展性以及灵活性。由于 C++ 继承了 C 的大部分语法，再加上本身具备复杂的类型系统以及泛型编程等语言特性，新人入门门槛也比较高。
3	Python	Python 语言发明于 1991 年，是一种解释型计算机编程语言，因此运行速度相对要慢。Python 除了支持面向对象编程之外，还支持函数式编程，它语法简单，更贴近人类自然语言，新人入门门槛较低。Python 是目前人工智能领域最热门的语言，对应的工具库非常丰富。

上面是以表格的形式来介绍 C、C++ 以及 Python 三种编程语言，但是在“介绍”那一列中的文本内容太长，我们可以换一种表达方式：

表 6-2 C vs C++ vs Python

C	C++	Python
由 AT&T 贝尔实验室发明于 1969 至 1973 年	由 Bjarne Stroustrup 发明于 1979 年	由 Guido van Rossum 发明于 1991 年
语法比较复杂，新人入门门槛高	语法比较复杂，新人入门门槛较高	语法简单，贴近人类自然语言，新人入门门槛低
编译型语言	编译型语言	解释型语言
支持面向过程编程	支持面向过程、面向对象编程	支持面向过程、面向对象、函数式编程
偏底层、运行速度快、使用灵活	继承了 C 语言的一些特性，在其基础之上还支持面向对象等特性	语法简单，学习难度低
一般用于驱动、编译器、嵌入式或者其他偏向硬件层面的开发	一般用于游戏前后端、PC 客户端的开发	一般用于数据科学、人工智能相关开发

上面表格一共还是 3 列，但是现在每列代表一种编程语言，列中的每个单元格是对该语言的描述，描述内容都比较精简。如果你想继续补充内容，可以对应地增加行即可。表格的组织方式多种多样，行可以变成列、列可以变成行，并没有严格的限制。我们只需要找一个适合自己的方式，比如上面这种每列代表一种语言，是因为该场景需要介绍的编程语言只有三种，如果数量再多点（或者数量不确定，后期会继续增加），那么表格宽度就不太够、这种组织方式就不再合适。

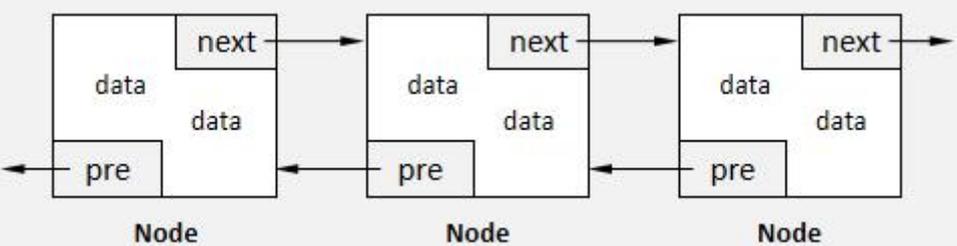
7 一图胜千言

人类在发明文字媒介之前，用的是图形符号。图像（或图形、图片）是所有内容表达方式中最直观的一种，同时也能提升读者的阅读兴趣。有人专门做过研究：在文档中增加图像能提升读者对文档的喜爱程度，不管这个图像跟文档内容本身是否有关系（[论文链接](#)）。也就是说，哪怕在文档中插入无关紧要的图像，读者也更愿意去尝试阅读文档中其他的内容。我们平时看别人演示 PPT 时，如果发现整页都是文字描述，大概率就不会有认真去听的欲望。下面是一段对双向链表的文字描述：

双向链表也叫双链表，是链表的一种。它的每个数据节点中都有两个指针，分别指向直接后继节点和直接前驱节点。所以，从双向链表中的任意一个节点开始，我们都可以很方便地访问它的前驱节点和后继节点。在应用双向链表时，我们一般构造双向循环链表，链表首尾相连。

上面这段描述双向链表的文字本身已经非常清晰，对数据结构有一定基础的人看完文字基本就能理解双向链表的结构和应用场合（基于它的特点）。但是，如果是一个零基础的小白来看这段话，可能效果就不会太好（尤其如果这段话是作为 PPT 中的内容，大概不会再有更多的内容补充）。如果我们在这段话后面增加一个插图，来直观告诉读者双向链表长什么样：

双向链表也叫双链表，是链表的一种。它的每个数据节点中都有两个指针，分别指向直接后继节点和直接前驱节点。所以，从双向链表中的任意一个节点开始，我们都可以很方便地访问它的前驱节点和后继节点。在应用双向链表时，我们一般构造双向循环链表，链表首尾相连。下图是双向链表结构示意图：



The diagram illustrates a doubly linked list structure with three nodes. Each node is represented as a rectangular box divided into three sections: a top-left section labeled 'data', a top-right section labeled 'next', and a bottom-left section labeled 'pre'. The 'next' field of one node points to the 'data' field of the next node to its right. Similarly, the 'pre' field of one node points to the 'data' field of the previous node to its left. The three nodes are labeled 'Node' below them. The first node's 'pre' field has an arrow pointing left, and the last node's 'next' field has an arrow pointing right, indicating a circular structure.

图 1 双向链表结构

上面的文本配合图片，能让读者更加直观的理解双向链表的结构特点。当文档中的文本和图片同时出现时，读者大概率会先看图片，然后再结合文字去理解，加快文档阅读速度。

7.1 可抽象也可具体

技术型文档中的插图不一定都得是流程图、架构图、或者结构设计图这种非常具体的技术相关图片，还可以是抽象的、能形象表达文档主题的图片。下面是在技术型文档中使用卡通和漫画图片的示例：

示例 1

Gitlab 中有 Label 和 Tag 两个概念。为了便于区分，这里将 Label 翻译成“标签”，将 Tag 翻译成“标记”（在有些地方这两个单词翻译并没有严格的差异）。

Gitlab 中标签的作用是为了分类、快速地检索和过滤，用户可以通过标签来直观的管理 Issues，比如 to-do、bug 等等。标记的主要作用是为了归档，给 Commit 取一个形象的别名，后期快速定位和查找。



Gitlab 中创建标记可以理解为“做记号”，建立索引。一般推荐为标记定义一个有意义的名称，比如以版本号为名，当我们要发布 1.0 版本，对应的标记名称可以是“v1.0”，如果我们要发布 2.0 预览版，那么对应的标记名称可以是“2.0-pre”。

示例 2

源码版本控制系统（Source Code Version Control System）主要负责对源代码版本进行管理，涉及到代码提交、撤销、比对、分支管理、代码合并等功能。源码管理是软件开发过程中非常重要的一个环节，它能有效保证软件代码质量。



图 1 团队协作

源码管理并不是软件开发周期的全部，整个软件开发周期涉及到多个流程、多个团队（多人）协作完成，包括立项/结项、进度/任务管理、需求/设计、bug 管理、测试、集成上线等环节。

7.2 突出图中重点

当我们想为文档添加图片时，单张图片包含的内容不宜太过复杂，图片应该能准确地表达意思。如果一张图太过复杂、或者包含了一些可能引起歧义的部分，我们可以尝试以下两种改进方式：

- ✓ 将复杂的图拆开，一张图对应一个局部细节；
- ✓ 在图片中将重点区域标记出来，让读者可以一眼就发现重点。

在技术型文档中插入复杂的系统架构图很常见，这种时候建议遵循“先宏观，再具体”的原则，循序渐进。我们不要一上来就放一张大图，还想将所有的细节都包含进去，这种想法不太现实，这不仅对你画图的技能要求很高，读者看完也容易一脸懵。下面这张图太过复杂：



整个视频分析系统由 3 大服务组成，分别是 Intelligent Video Analytics、Front Video Service 以及 Distribute Load Balance，这 3 大服务一共包含 15 个子模块。下面是视频分析系统结构：

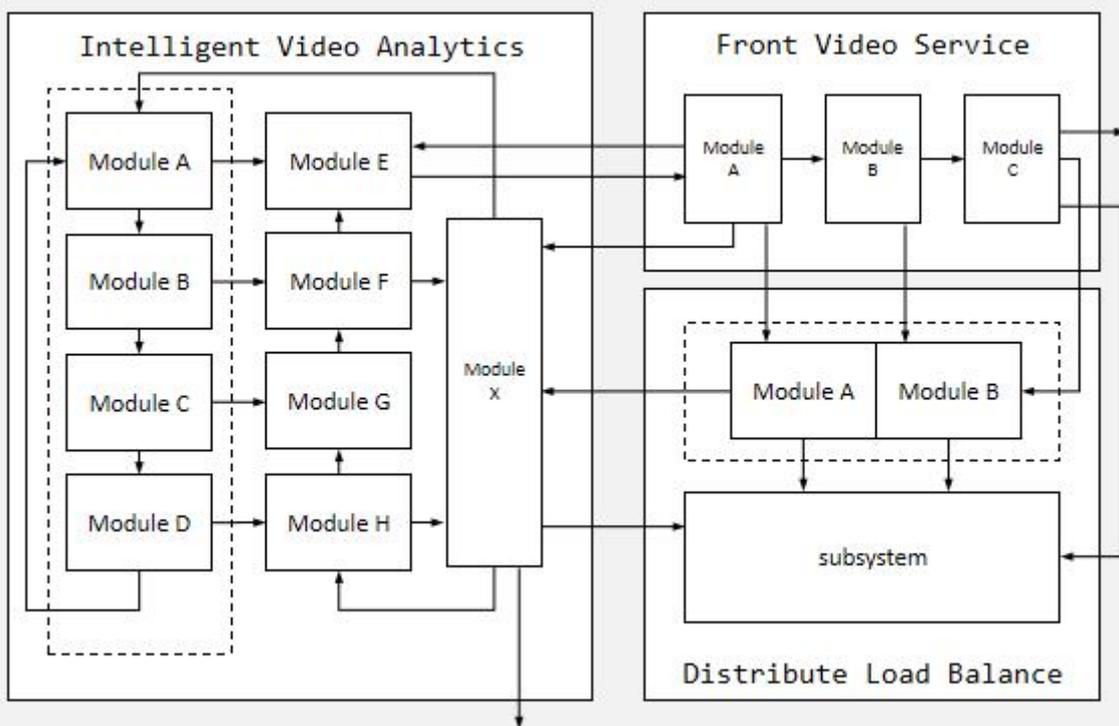


图 1 视频分析系统结构

上面这个例子中插入的这张图既想描述 3 大服务之间的交互关系、又想描述各个服务内部子模块之间的交互关系（上面只是示意图，实际情况可能比这个更复杂）。文档读者碰到这种

情况可能会产生两个感觉：一是图太复杂了，很难看懂，有些地方迫于空间原因字号还小；二是我需要重点关注的点在哪里？如果遵循前面提到的“先宏观，再具体”的原则，上面这个例子可以调整为：



整个视频分析系统由 3 大服务组成，分别是 Intelligent Video Analytics、Front Video Service 以及 Distribute Load Balance。下面是视频分析系统中各服务之间的关系：

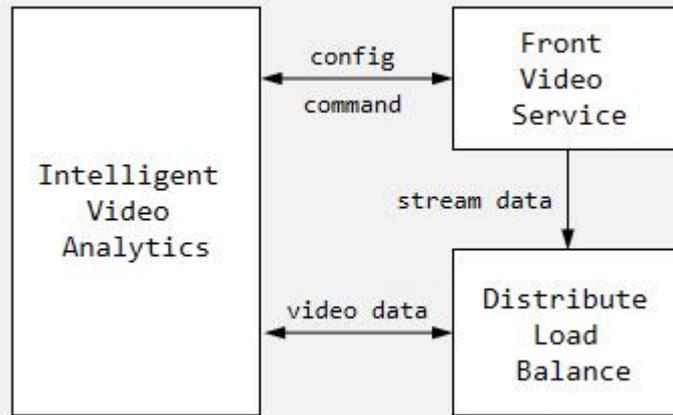


图 1 视频分析系统服务交互

其中，Intelligent Video Analytics 服务主要负责对视频解码、推理以及行为分析等结构化操作。该服务内部一共包含 9 个子模块，模块之间的关系见下图：

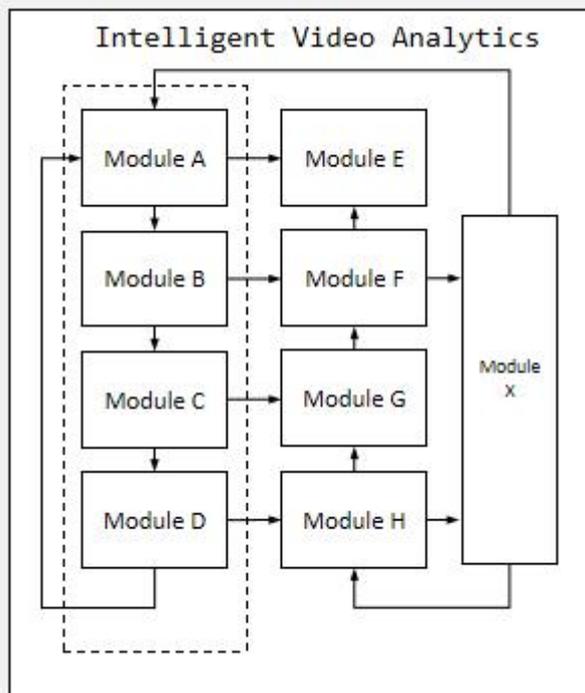


图 2 Intelligent Video Analytics 服务子模块交互

Front Video Service 服务主要负责视频接入、分发、配置管理等功能。该服务内部一共包含 3 个子模块...

另外一种情况，插入的图片中包含了不相干内容，文档作者又没有给出醒目的标记，读者看完不清楚关注重点在哪里。下面是错误的示例：



Gitlab 中的 Release 功能主要用来对仓库中的代码以及其他一些相关资料文件进行归档，通常用于版本发布。当有新版本发布时，用户可以基于对应的 Commit 创建一个 Tag 标记，给它一个合理的名字，比如“v1.0-pre”（代表发布 1.0 预览版），然后再基于该 Tag 发布版本。后期，其他人可以通过 Release 菜单快速浏览、检索项目版本发布记录以及对应时间点的相关代码和资料。用户可以在 Gitlab 主界面的左侧菜单中找到 Release 功能入口：

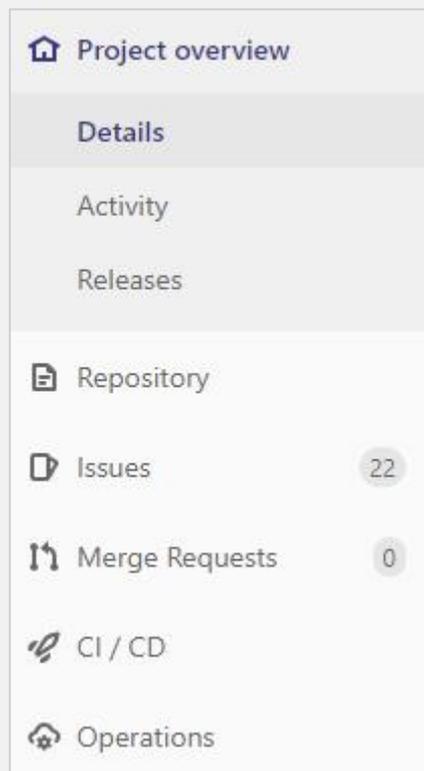


图 1 Gitlab 中 Release 菜单

上面图片在介绍 Release 功能时给出的图片中包含的菜单项太多，为了让读者更直观看懂图片关注点，可以将图片调整如下（左右两种都可以）：



Gitlab 中的 Release 功能主要用来对仓库中的代码以及其他一些相关资料文件进行归档，通常用于版本发布。当有新版本发布时，用户可以基于对应的 Commit 创建一个 Tag 标记，给它一个合

理的名字，比如“v1.0-pre”（代表发布 1.0 预览版），然后再基于该 Tag 发布版本。后期，其他人可以通过 Release 菜单快速浏览、检索项目版本发布记录以及对应时间点的相关代码和资料。用户可以在 Gitlab 主界面的左侧菜单中找到 Release 功能入口：

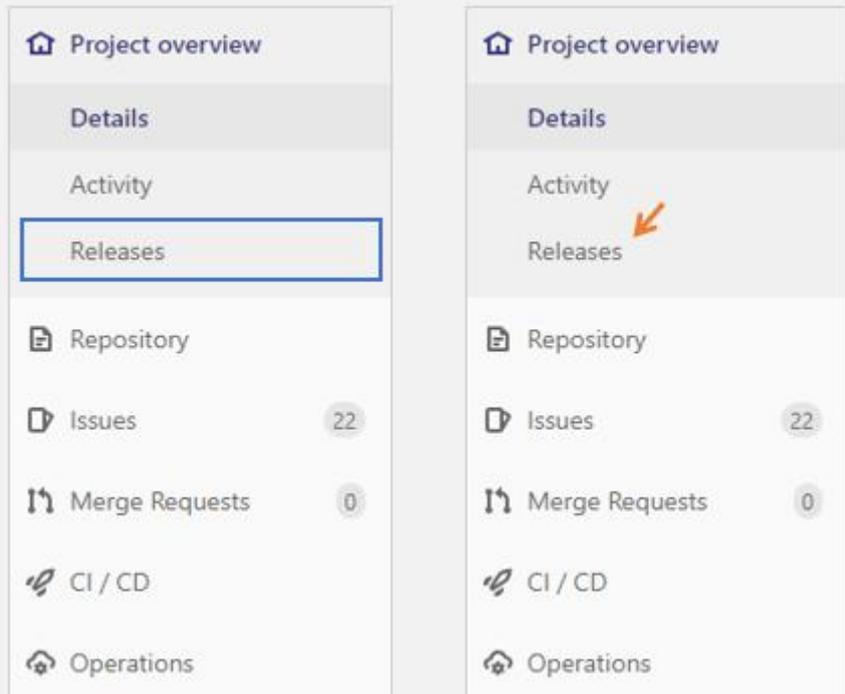


图 1 Gitlab 中 Release 菜单

7.3 有准确的图标题

图片是为了读者能够更直观地理解文档内容，但是图片毕竟不是文字，不同的人对同一张图片理解可能存在差异，尤其对于那种不包含任何文字的图片。因此，在文档中插入任何图片时，我们应该为它定义一个合适、贴切的标题。图标题一般是一个名词或者短语，作用跟前面讲到的表格标题一样，协助读者理解图片所要表达的含义。

8 统一样式和风格

文档的样式和风格其实跟我们写代码一样，写代码要遵守统一的代码风格（变量命名、换行规则等等），写文档也应该遵守统一的文档风格。公司或者组织一般都有自己的文档风格规

范，规范会定义好正文/标题字体字号、页眉页脚、页边距、行间距、段前段后间距等等，按照规范写出来的文档风格基本就能保持一致。

对于没有规范可用的场合，文档作者可以根据自己的偏好执行即可，保证整篇文档的内容遵守相同的风格，比如文档开头和文档结尾的段落间距、列表样式、对齐方式都应该保持一致。本篇文档的主要规范定义如下：

-
- ✓ 页边距上下左右 2cm；
 - ✓ 标题 18 号华文仿宋，正文 12 号宋体，正文中表格/图标题 12 号华文仿宋；
 - ✓ 段前/段后间距 0.5，段落行间距 1.5 倍，段落首行对齐不空格；
 - ✓ 表格、图片居中对齐，图标题在图片下方、表格标题在表格上方。
-

还有另外一些比较重要的样式定义，比如列表样式（本篇文档中每个列表外面套了一个表格，表格无左右边框），还比如本篇文档涉及到了很多举例和示范，所有的举例示范都在表格中，并且表格有自己的样式（字体字号、背景颜色等等）。

9 把握好整体文档结构

把握好整体文档结构是一件非常困难的事情，这个其实跟前面讲到的文档内容本身没什么关系。文档作者在动笔之前需要有一个宏观的构思，需要在脑子里先将文档大纲梳理一遍，一级标题可以是什么、二级标题又可以是什么，然后考虑将合适的内容放到对应的章节中去。优秀的作者在正式动手之前，可能已经有了很长一段时间的思考准备，尤其对于那种非常复杂的文档。但是这种方式对一些人来讲可能不太现实，难度太大。那么这时候就只能考虑另外一种方式，动手之前先在白纸上打草稿，列出来文档大纲，然后不断修改和调整，直到满意为止。

其实不管上面哪种方式，文档结构考验的是作者组织内容的思维能力。对于一些需求、设计类型的“主流”技术型文档，考验的是作者对软件需求、系统架构的理解深度，该写什么不该写什么，写到什么程度，这些都需要作者考虑清楚，这类型的文档一般有标准的模板可以参考，大家平时写得/见得也比较多。对于另外一些“非主流”类型的技术型文档，比如对某

个线上问题的分析报告、技术/原型调研类文档，这些文档一般规模比较小、也没什么参考标准，全靠作者自己去组织。

下面就以“对某个用户需求做技术性反馈”为例，抛砖引玉，简单描述一下技术型文档结构应该如何去组织：

场景说明

视频分析系统中，客户要求的事件录像文件对涉事车辆目标（或区域）进行高亮标框显示，视频录像在播放时会有一个醒目的多边形提醒用户具体事件发生位置。客户懂一些技术相关知识，要求公司技术研发团队针对该需求给出合理的需求反馈，如果需求可实现，评估工作难度；如果需求不可实现，说明具体原因。

根据上面场景说明，该需求并非硬性要求。甲方提出了一个想法，并且非常贴心地考虑到了乙方是否具备条件实现，希望给出一个实质性的答复。公司技术团队在写反馈说明文档之前，应该考虑以下两个问题：

- ✓ 如果正常响应该需求，具体的方案是什么、难点/风险点各是什么；
- ✓ 如果不能正常响应该需求，具体原因是什么，是否有可替代方案、替代方案是什么。

也就是说，不管最终团队是否响应该需求，我们在文档中都要有非常实质性的内容，不应该是空话、套话。下面就以“不响应”为例，描述文档应该包含哪些内容：

序号	节标题名称	主要内容
1	背景说明	用自己的话将客户的需求完整描述一遍，不要有任何偏差，表明我方 已认真理解过原始需求 。
2	已有录像逻辑	详细描述系统中目前已有的事件录像逻辑。因为我们本次是不响应该需求，所以对后面不响应有利的内容一定要着重强调（ 要突出已有录像逻辑的优势 ）。
3	录像标框逻辑	详细描述在事件录像文件对涉事目标（或区域）进行高亮标框的逻辑。注意这里按照理想逻辑去描述， 不用考虑任何外在限制 。
4	录像标框难点	结合第 3 点， 重点归纳、整理出在录像文件中标框的难点 ，比如需要对每一路进行解码再去叠加图形、视频画面不能压缩否则影响分辨率等等，这些对设备性能要求非常高，会增加硬件成本。

序号	节标题名称	主要内容
5	解决方案一 (不计代价去响应)	按照理想逻辑去响应,但是要提出前提条件或者代价,比如单台设备分析路数降低到原来的一半,硬件成本是原来的2本。 (其实就是要排除这个方案)
6	解决方案二 (可替代方案)	提出一种可替代的方案,可以满足客户最开始提出的“有醒目标记提醒用户”。比如当视频录像播放时,可以在播放器上面 叠加 一个高亮方框,能够 大概 标记涉事车辆目标(或区域)。同时,强调该方案的优势(比如工作周期短、对成本无影响)。
7	结论	其实根据前面的描述,只要认真读完文档的人基本都能知道结论是什么、应该选哪个方案。但是这里还是 要书面写上 ,根据前面的描述,解决方案二有更大的优势,建议采用方案二。

需要注意的是,“响应”或者“不响应”的决定很多时候不在技术团队或者写这个文档的人手里。虽然文档中的内容应该为最终的结论服务,但是总体上不应该有偏差。

10 明确文档的目标群体

文档的目标群体是谁?这个其实应该是写文档最开始就需要明确的东西,面对不同的群体,我们文档的内容、结构包括内容描述程度都会不同。尽早确定读者有助于在构思阶段就明确文档内容边界,哪些该写、哪些不该写,该写的又应该如何去写,这些都是编写文档的大方向。