



# AI 编程之前的那些事儿

## All You Need before Vibe Coding

---

AI 写代码，你心里得有数  
编程是一种修行，要知行合一

作者：阿色

( AI 辅助编写 )

2026.5 @大庆

大系统观开放论坛

[www.holomind.com.cn](http://www.holomind.com.cn)

送给 AI 时代怀揣梦想的人

个人英雄主义的时代又回来了

加油吧，梦想家！

## 作者简介

阿色，网名阿色树新风（ArthurTreeNewBee），真名王权，1968年生，大庆油田资深老程序猿，本地区第一个软件著作权获得者，教授级正高级工程师。

1987年开始写代码，从 Fortran 起步，近四十年断断续续，最近重新上路。

1999年提出"数字油田"概念，并不懈坚定践行。

2016年完成开源著作《大系统观》。

现在还在编程，还在学习，还在折腾 AI。

# 前言

## 这本书写给谁？

### 一位依然怀揣梦想的老哥

前段时间刷抖音，阿色看到有个老哥，看样子他年纪应该比我大，应该是老哥。

他找到一个程序员说：

你帮我个小忙。

我花了一个月时间，开发了一个非常有用的 A 批批，能造福全人类。

程序都编完了，在豆包里。

但是我不知道怎么运行，你帮我一下。

注意，他用了一个“小”字，他觉得这个事情很容易。

那个程序员，瞬间石化了！

阿色看后哈哈大笑。然后，这事就过去了。

### 一位初生牛犊不怕虎的小哥

又过了几天，2026 年 5 月 10 日周日上午 9 点左右，我躺在家里沙发上刷手机。

是的，阿色也天天刷手机。阿色认为刷手机没什么不好，除了对眼睛不好。

关键在于你刷的是什么？现在很多新媒体的内容比纸质书更方便、更新鲜，也越来越系统。

OK，阿色刷到一个很有梦想的小伙子，他正在用 AI 编程序。

但是，他不懂 AI 和 IT 的基础知识，程序怎么都弄不过去，非常烦恼。

我突发一念，我得给这个孩子写本书。

这本书也顺便写给那位老哥吧。

## 那些想跟 AI 做朋友的朋友们

So，这本书是写给：

- 那些想用 AI 写代码，但又不太懂计算机基础的普通朋友。
- 那些曾经会用 AI 写代码，但在新时代掉队了的老程序猿朋友。
- 那些遇到报错就懵圈，只会反复问 AI 的朋友。
- 那些想学点计算机基础，但不想看枯燥教材的朋友。

## 这本书讲什么？

讲那些你在 AI 编程之前应该知道的事儿——计算机是啥、程序怎么跑、网络咋连的、数据咋存的、报错咋看、问题咋查.....

不讲高深的，就讲基础的。基础打牢了，AI 编程才能玩得转。

**最重要的：帮你建立编程思维。**

## 这本书啥风格？

简单、幽默、大白话，东北味儿，看着不累。

没有公式推导，没有术语堆砌，就像跟你唠嗑一样，但是，不太严谨，请注意。

## 这本书有啥用？

看完这本书，你用 AI 写代码的时候就能心里有数。

AI 给的代码，你能看懂；AI 报的错，你能排查；AI 给的方案，你能判断好不好。

## 这本书是咋写出来的？

这本书是在 AI 的辅助下写出来的。

在我决定给那个小伙子写一本书之后，我打开微信，说了二三百字，描述一下我想写一本什么样的书，然后发给 AI（QwenPaw + GLM5）——写书开始了！

AI 马上列了一个提纲。

我对提纲提了一些要求。

AI 先写了初稿，用了大约 10 分钟，写了大约 4 万字，Markdown 格式。

我让他转成 Word 格式发给我。

2 分钟后我收到了 Word 格式的文档。

我用 10 分钟左右浏览了这个文档。

我又提出了一些要求。

AI 重新编写。

10 分钟后，新版出来了，就是现在你看到的这个 Word 文档。

上面我与 AI 的整个互动都是通过微信进行的，直到现在我用电脑对这个 Word 文件进行修改和审定。

我在电脑上一节一节看、改，有 AI 帮我润色，听着我让 AI 生成的歌曲《天堂的密码》，还有老同学春阳大哥用 AI 创作的《听诊大地的人》。

## 这本书需要仔细看么？

阿色认为，不需要。

当然如果你想仔细看，也没人拦着你。

这本书是个入门扫盲读物，不是教材，不是参考书，也不是速成秘籍。

阿色先要提醒你，这本书可能存在细节上的错误。一是，AI 生成的东西不能保证全部正确；二是，阿色对很多新东西也已经不太懂了，不能保证完全纠正 AI 的错误。

但是，大系统观讲全息思维，局部的小问题不影响系统的整体功能。

建议你先浏览一遍再说。

你可以：

- 通读一遍，对计算机基础知识有个整体了解

- 遇到不懂的概念，问问豆包
- 结合 AI 编程实践，边写代码边理解原理
- 推荐给同样想学 AI 编程的朋友

你不应该：

- 指望看完就能成为编程高手
- 把它当教材，里面没有考试重点
- 期望学到具体的编程技巧，那些得自己练，这个只是前奏

## 再唠两句儿

这本书，是一个老程序员的碎碎念。

我写得不深，但都是我觉得重要的东西。

我写得比较白话，因为我觉得技术不应该只有专家能懂。

我写得带点东北味儿，因为我是东北人，这么写比较自然。

如果这本书能帮你少走点弯路，能让你用 AI 写代码的时候更有底气，那就值了。

来吧，咱们开始。

# 目 录

前言 .....	I
这本书写给谁? .....	I
一位依然怀揣梦想的老哥 .....	I
一位初生牛犊不怕虎的小哥 .....	I
那些想跟 AI 做朋友的朋友们 .....	II
这本书是咋写出来的? .....	II
这本书需要仔细看么? .....	III
再唠两句儿 .....	IV
第一章：计算机软硬件的那点事儿.....	1
开场白 .....	1
1.1 计算机能干啥，不能干啥? .....	1
1、计算机不是神仙 .....	1
2、计算机能干啥? .....	2
3、计算机不能干啥? .....	2
4、人工智能也是计算 .....	3
小结一下 .....	3
1.2 CPU、内存、硬盘、GPU——都是干啥的? .....	4
1、CPU：大脑 .....	4
2、内存：工作台 .....	5



3、硬盘：仓库	6
4、GPU：原本是个画画的高手，现在变成了人工智能大力士	7
5、哥四个咋配合？	8
小结一下	8
<b>1.3 操作系统：计算机的大管家.....</b>	<b>9</b>
1、操作系统是干啥的？	9
2、操作系统管啥？	9
3、常见的操作系统	10
4、选哪个操作系统？	12
小结一下	12
<b>1.4 软件是咋回事儿？ .....</b>	<b>12</b>
1、软件是啥？	12
2、软件分几种？	13
3、软件是咋来的？	13
4、软件的运行	14
5、为啥有的软件装不上？	14
小结一下	15
<b>1.5 计算机咋跟人交流：输入输出那些事儿.....</b>	<b>15</b>
1、人咋跟计算机说话？	15
2、计算机咋跟人说话？	15
3、标准输入输出	16
5、重定向	16
小结一下	17

本章小结.....	17
<b>第二章：编程序的那点事儿.....</b>	<b>19</b>
开场白 .....	19
<b>2.1 编程语言是个大家族.....</b>	<b>19</b>
1、啥是编程语言？	19
2、编程语言的演变	20
3、高级语言有啥好处？	20
4、常见的高级语言	21
5、咋选语言？	22
小结一下	23
<b>2.2 代码咋变成行动的？ .....</b>	<b>23</b>
1、写完代码就能跑吗？	23
2、编译型语言	23
3、解释型语言	24
4、中间派：Java	24
5、你的 AI 代码咋运行？	25
小结一下	25
<b>2.3 变量、条件、循环、函数——编程就其实那么几下子.....</b>	<b>26</b>
1、编程的核心概念	26
2、变量：给数据起个名字	26
3、数据类型：数据有啥种类	27
4、条件：让程序有判断力	28
5、循环：让程序能重复干活	30

6、函数：打包一段功能	31
小结一下	33
<b>2.4 程序是咋组织的？ .....</b>	<b>33</b>
1、代码文件	33
2、模块和包	34
3、库和框架	34
小结一下	35
本章小结.....	36
<b>第三章：数据、库和算法的那点事儿 .....</b>	<b>37</b>
开场白 .....	37
<b>3.1 啥都能变成数字！ .....</b>	<b>37</b>
1、计算机的世界观	37
2、二进制：计算机的语言	38
3、文字咋变成数字？	39
4、图片咋变成数字？	40
5、声音咋变成数字？	41
6、视频咋变成数字？	42
小结一下	42
<b>3.2 文件格式那点秘密 .....</b>	<b>43</b>
1、文件后缀名是干啥的？	43
2、常见文件格式	43
3、压缩文件	45
4、文件格式选择建议	45

小结一下	46
<b>3.3 数据库：比 Excel 高级的大表格 .....</b>	<b>46</b>
1、啥是数据库？	46
2、为啥需要数据库？	46
3、关系型数据库	47
4、SQL：和数据库对话的语言	48
5、常见的关系型数据库	49
6、NoSQL 数据库	49
7、JSON 和数据交换格式	51
小结一下	51
<b>3.4 数据结构和算法：程序员的内功 .....</b>	<b>51</b>
1、啥是数据结构？	51
2、啥是算法？	53
3、算法复杂度	54
4、你需要学吗？	55
小结一下	55
<b>本章小结 .....</b>	<b>55</b>
<b>第四章：网络的那点事儿 .....</b>	<b>57</b>
开场白 .....	57
<b>4.1 两台电脑咋说话？ .....</b>	<b>57</b>
1、网络的本质	57
2、网络协议：电脑的"语言"	57
3、数据咋传输？	58

小结一下	59
<b>4.2 IP 地址：网络世界的门牌号</b>	<b>59</b>
1、啥是 IP 地址？	59
2、IP 地址长啥样？	59
3、IPv4 和 IPv6	60
4、公网 IP 和内网 IP	60
5、怎么查自己的 IP？	61
小结一下	61
<b>4.3 域名和 DNS：好记的名字</b>	<b>61</b>
1、域名是啥？	61
2、域名的结构	61
3、DNS：域名翻译官	62
4、hosts 文件	63
小结一下	63
<b>4.4 网站是咋打开的？</b>	<b>63</b>
1、浏览器访问网站的过程	63
2、三次握手：建立连接	64
3、HTTP 请求和响应	64
4、HTTP 和 HTTPS	65
小结一下	66
<b>4.5 常见网络概念扫盲</b>	<b>66</b>
1、端口：一栋楼的房间号	66
2、URL：统一资源定位器	67

3、防火墙：网络安全	67
4、代理服务器：中间人	68
5、VPN：虚拟专用网络	69
6、CDN：内容分发网络	69
小结一下	70
本章小结.....	70
<b>第五章：网页和前端的那点事儿.....</b>	<b>71</b>
开场白 .....	71
<b>5.1 网页的三驾马车.....</b>	<b>71</b>
1、HTML、CSS、JavaScript	71
2、HTML：搭骨架	71
3、CSS：穿衣服	73
4、JavaScript：让网页动起来	74
5、三者怎么配合？	74
小结一下	76
<b>5.2 前端和后端是啥关系？ .....</b>	<b>76</b>
1、啥是前端？	76
2、啥是后端？	76
3、前后端怎么配合？	77
4、打个比方	77
5、前后端分离	78
6、AI 编程时代的前后端	78
小结一下	79

<b>5.3 API 接口：前后端的传话筒 .....</b>	<b>79</b>
1、啥是 API?	79
2、接口长啥样?	79
3、常见的请求方法	80
4、接口文档	81
5、怎么调用接口?	82
小结一下	83
<b>5.4 前端框架：别再手写原生代码了 .....</b>	<b>83</b>
1、为啥要框架?	83
2、三大前端框架	84
3、框架的好处	85
4、选哪个框架?	85
小结一下	86
<b>5.5 用 AI 写前端代码 .....</b>	<b>86</b>
1、AI 能帮你写啥?	86
2、怎么让 AI 给你写好代码?	88
3、小心 AI 的坑	89
小结一下	89
<b>本章小结 .....</b>	<b>90</b>
<b>第六章：服务器和后端的那点事儿 .....</b>	<b>91</b>
开场白 .....	91
<b>6.1 后端是干啥的? .....</b>	<b>91</b>
1、后端的职责	91

2、后端处理的流程	92
小结一下	92
<b>6.2 后端语言哪家强? .....</b>	<b>92</b>
1、常见的后端语言	92
2、Python: 简单就是美	93
3、Java: 稳就是赢	93
4、Go: 快就是强	94
5、Node.js: 前端也能干后端	95
6、选哪个?	96
小结一下	96
<b>6.3 Web 框架: 造轮子不如用轮子 .....</b>	<b>97</b>
1、啥是 Web 框架?	97
2、常见的 Web 框架	97
3、Django: 大而全	97
4、Flask: 小而精	98
5、FastAPI: 又快又好	98
小结一下	99
<b>6.4 数据库操作: ORM 还是 SQL? .....</b>	<b>99</b>
1、啥是 ORM?	99
2、ORM 的好处	100
3、ORM 的缺点	100
4、常见的 ORM	101
5、建议	101



小结一下	101
<b>6.5 用户认证：你是谁，能干啥？</b>	<b>102</b>
1、啥是用户认证？	102
2、传统的认证方式：Session	102
3、现代的认证方式：Token	102
小结一下	103
本章小结	103
<b>第七章：云、部署和运维的那点事儿</b>	<b>105</b>
开场白	105
<b>7.1 "上云"不是真上天</b>	<b>105</b>
1、啥是云计算？	105
2、云服务商有哪些？	106
3、云服务的三种类型	106
4、云服务的计费方式	107
小结一下	107
<b>7.2 服务器：程序的安身之处</b>	<b>108</b>
1、啥是服务器？	108
2、服务器的类型	108
3、虚拟服务器和云服务器	109
4、服务器的操作系统	109
小结一下	109
<b>7.3 部署：把程序送上战场</b>	<b>110</b>
1、啥是部署？	110

2、部署的基本步骤	110
3、连接服务器	110
4、上传代码	111
5、启动程序	111
6、域名和 HTTPS	112
小结一下	113
<b>7.4 Docker：把环境打包带走.....</b>	<b>114</b>
1、环境不一致的烦恼	114
2、Docker 是啥？	114
3、Docker 的基本使用	115
4、Docker 的好处	115
小结一下	116
<b>7.5 运维：让程序稳定跑下去.....</b>	<b>116</b>
1、运维是干啥的？	116
2、运维的两种方式	116
3、运维的核心任务	117
4、运维工具推荐	118
5、宝塔面板：运维神器	118
<b>本章小结.....</b>	<b>120</b>
<b>第八章：AI 和大语言模型的那点事儿.....</b>	<b>121</b>
<b>开场白.....</b>	<b>121</b>
<b>8.1 AI 是个啥？跟传统程序有啥不同？.....</b>	<b>121</b>
1、啥是人工智能？	121

2、图灵测试：你怎么知道对面是人是机器？	122
3、AI 和传统程序的区别	122
4、机器学习和深度学习	123
小结一下	123
<b>8.2 大语言模型是咋工作的？ .....</b>	<b>123</b>
1、大语言模型的“大”是啥意思？	123
2、预测下一个词的核心玩法	124
3、训练和推理	124
4、Token 是啥玩意儿？	125
5、幻觉：AI 为啥会胡说八道？	126
小结一下	126
<b>8.3 常见的大语言模型 .....</b>	<b>126</b>
1、国外的选手	126
2、国内的选手	127
3、选哪个好？	128
小结一下	129
<b>8.4 跟 AI 聊天：提示词那点学问 .....</b>	<b>129</b>
1、啥是提示词？	129
2、写提示词的基本技巧	129
3、进阶技巧	130
4、别把 AI 当神仙	130
小结一下	130
<b>8.5 AI 编程的正确姿势 .....</b>	<b>131</b>

1、AI 能帮咱干啥？	131
2、AI 不能干啥？	131
3、怎么用好 AI 编程？	131
4、AI 编程的未来	132
小结一下	133
<b>8.6 OpenClaw 及各种龙虾 .....</b>	<b>133</b>
1、什么是龙虾？	133
2、其他龙虾产品	133
3、龙虾适不适合做 AI 编程？	134
4、编程该怎么选工具？	134
本章小结 .....	135
<b>第九章：编程工具的那点事儿 .....</b>	<b>135</b>
开场白 .....	135
<b>9.1 编程环境是啥玩意儿？ .....</b>	<b>136</b>
1、啥是编程环境？	136
2、环境咋配？	136
3、环境变量是啥？	137
小结一下	137
<b>9.2 代码编辑器：VS Code 为啥这么火？ .....</b>	<b>138</b>
1、啥是代码编辑器？	138
2、常见编辑器	138
3、IDE 和编辑器的区别	138
4、VS Code 为啥这么火？	139

5、怎么用 VS Code?	139
小结一下	140
<b>9.3 终端和命令行：别怕黑窗口 .....</b>	<b>140</b>
1、啥是终端?	140
2、打开终端	140
3、常用命令	141
4、路径是啥?	141
5、小技巧	142
小结一下	142
<b>9.4 Git：版本控制的王者 .....</b>	<b>142</b>
1、啥是版本控制?	142
2、Git 是啥?	143
3、Git 的基本操作	143
4、Git 的工作流程	144
5、GitHub 是啥?	145
小结一下	145
<b>9.5 AI 编程助手：你的随身参谋 .....</b>	<b>145</b>
1、啥是 AI 编程助手?	145
2、国外 AI 编程助手	146
3、国内 AI 编程助手	146
4、怎么用 AI 编程助手?	147
5、使用建议	147
小结一下	148

本章小结.....	148
<b>第十章：与 bug 作斗争的那点事儿.....</b>	<b>149</b>
开场白 .....	149
<b>10.1 报错信息咋看？ .....</b>	<b>149</b>
1、报错信息是啥？ .....	149
2、常见的错误类型 .....	149
3、怎么看报错信息？ .....	150
4、实例：看一个 Python 报错 .....	151
小结一下 .....	151
<b>10.2 常见错误类型 .....</b>	<b>151</b>
1、TypeError： 类型错误 .....	151
2、NameError： 名字错误 .....	152
3、IndexError： 索引错误 .....	152
4、KeyError： 键错误 .....	152
5、FileNotFoundError： 文件不存在错误 .....	152
6、AttributeError： 属性错误 .....	153
7、IndentationError： 缩进错误 .....	153
小结一下 .....	153
<b>10.3 Debug 的基本招数.....</b>	<b>154</b>
1、啥是 Debug？ .....	154
2、打印调试 .....	154
3、用调试器 .....	154
4、二分法定位 .....	155

5、注释法	155
6、搜索错误信息	155
小结一下	155
<b>10.4 怎么问 AI? .....</b>	<b>156</b>
1、问 AI 的姿势	156
2、错误的问法	156
3、正确的问法	156
4、问 AI 的技巧	157
5、不要完全依赖 AI	157
小结一下	157
<b>10.5 求助的正确姿势.....</b>	<b>158</b>
1、啥时候该求助?	158
2、去哪求助?	158
3、怎么提问?	158
4、提问的礼仪	159
小结一下	159
<b>本章小结.....</b>	<b>160</b>
<b>第十一章：安全和法律的那点事儿.....</b>	<b>161</b>
<b>开场白 .....</b>	<b>161</b>
<b>11.1 代码安全基本意识.....</b>	<b>161</b>
1、啥是代码安全?	161
2、常见的安全问题	161
3、安全编码的基本原则	162

4、AI 写的代码安全吗？	163
小结一下	163
<b>11.2 保护你的代码和数据 .....</b>	<b>163</b>
1、代码要保护啥？	163
2、怎么保护代码？	163
3、数据要保护啥？	164
4、怎么保护数据？	164
5、云服务安全	164
小结一下	165
<b>11.3 开源协议和版权 .....</b>	<b>165</b>
1、啥是开源协议？	165
2、常见的开源协议	165
3、怎么选开源协议？	166
4、用开源代码要注意啥？	166
小结一下	167
<b>11.4 AI 编程的法律风险 .....</b>	<b>167</b>
1、AI 生成内容的版权问题	167
2、AI 可能"抄"别人的代码	167
3、怎么降低风险？	167
4、AI 使用的合规问题	168
5、AI 服务的隐私问题	168
小结一下	168
<b>11.5 职业道德和责任 .....</b>	<b>168</b>



1、程序员的职业道德	168
2、AI 编程的道德问题	169
3、出了问题谁负责？	169
小结一下	169
<b>本章小结.....</b>	<b>170</b>
<b>附        录.....</b>	<b>171</b>
<b>附录 A：常用术语表（中英文对照）.....</b>	<b>171</b>
一、计算机基础	171
二、编程语言	171
三、数据类型与结构	172
四、程序结构	172
五、软件工程	173
六、网络与 Web	173
七、数据库	174
八、云计算与部署	174
九、调试与错误	175
十、安全与法律	175
十一、AI 与机器学习	176
十二、其他常用术语	176
<b>附录 B：推荐学习资源.....</b>	<b>177</b>
在线教程	177
视频课程	177
练习平台	177

社区 .....	177
附录 C: Windows CMD/PowerShell 常用命令 .....	178
附录 D: VS Code 常用快捷键 .....	178
后记 .....	180
写这本书的初衷 .....	180
关于 AI 编程的一点思考 .....	181
给读者的话 .....	181
感谢 .....	182
最后 .....	182

# 第一章：计算机软硬件的那点事儿

## 开场白

在开始聊 AI 编程之前，咱们先得搞清楚一个最基本的问题：计算机这玩意儿，到底是啥？

计算机这个词的含义比电脑更宽一些，电脑一般是指个人用的桌面计算机。除电脑之外，计算机还包括其他类型的计算机，比如，大型计算机、服务器、手机，甚至量子计算机，等等很多。

你天天用电脑、用手机，但你知道它里面是咋工作的吗？你知道为啥它能算数、能放视频、能打游戏吗？

这一章，咱们就用大白话唠唠计算机的基本原理。不整那些复杂的电路图和公式，就讲你能听懂的东西。

理解了计算机的基本原理，你才能理解程序是咋回事，才能理解 AI 给你的代码在干啥。

## 1.1 计算机能干啥，不能干啥？

### 1、计算机不是神仙

先说个大实话：计算机不是神仙，它就是个工具。

很多人觉得计算机很聪明，能干很多事儿。其实计算机一点都不聪明，它只会做一件事儿：算数。

加减乘除，0 和 1，就这些。

但它算得特别快，快到你感觉不到它在算。

**举个例子：**

你看视频，计算机在干啥？在算。把视频文件里的数据解码成图像和声音。

你打游戏，计算机在干啥？在算。算画面咋显示、人物咋动、碰撞咋处理。

你上网，计算机在干啥？还在算。算数据咋传输、网页咋渲染。

你用 AI 写代码，计算机在干啥？还是在算。算你输入的文字、算模型的参数、算输出的代码。

所以，计算机本质上就是个"算得快的铁盒子"。

## **2、计算机能干啥？**

计算机能干的事儿，都是能转化成计算的事儿，能算的它都能干：

### **存储信息**

把数据变成 0 和 1 存起来。你存的文档、照片、视频，在计算机眼里都是 0 和 1。

### **处理数据**

对数据进行各种计算。加加减减、比较大小、逻辑判断，都是计算。

### **显示图像**

把数据变成图像输出到屏幕。每个像素点的颜色，都是计算出来的。

### **播放声音**

把数据变成声音输出到扬声器。每个时刻的音量、频率，都是计算出来的。

### **联网通信**

把数据通过网络传给别人。怎么打包、怎么传输、怎么校验，都是计算。

这些事儿，说到底都是算。

## **3、计算机不能干啥？**

计算机不能干的事儿，是那些没法转化成计算的事儿。

在没有大语言模型之前，计算机肯定干不了下面的事情，但现在能不能有点不好

说，如果通用人工智能实现了，下面这些就能干了。暂且还是认为不能吧。

### 不能创造

计算机不会"发明"东西，它只能按你给的规则处理数据。

你给它一堆数据，它能帮你分析、总结、预测。但它不会凭空创造出新东西。

有人说 AI 不是能创作吗？能写文章、能画图、能编代码吗？

是的，但 AI 本质上也是在算。它看了大量数据，学到了规律，然后按规律生成新东西。它不知道自己在创作，只是在执行算法。

### 不能理解

计算机不知道"意义"是啥，它只是在处理符号。

你让它翻译"我爱你"，它把这三个字翻译成"I love you"。但它不知道"爱"是啥意思，不知道这句话背后的情感，只是在做符号替换。

### 不能判断

计算机不会"觉得"好坏对错，它只是按条件执行。

你给它一个判断条件，它返回 **True** 或 **False**。但它不知道这个判断有什么意义，只是在执行逻辑运算。

## 4、人工智能也是计算

没有现代计算机技术，人工智能是不可能实现的，因为人工智能也是基于计算的。

从几十年前开始的最简单的判断式人工智能，到现在基于大语言模型的人工智能，计算都是它的基础。

### 小结一下

- 计算机不是神仙，它就是个算得快的工具
- 计算机能干的事，都是能转化成计算的事
- 计算机不能干的事，是那些没法转化成计算的事
- 用计算机，得懂它、会用它
- 人工智能是基于计算的

## 1.2 CPU、内存、硬盘、GPU——都是干啥的？

计算机的四大件儿：

打开一台计算机，里面有很多零件。但最核心的，基本就四个：CPU、内存、硬盘、GPU（可以没有）。

这哥四个各有分工，配合工作，计算机才能跑起来。

咱们一个一个唠。

### 1、CPU：大脑

CPU，全叫"中央处理器"，你可以把它理解为计算机的大脑。

CPU 干啥活儿？

它负责执行指令、进行计算、控制其他零件干活。

你打开一个软件，CPU 读取软件的指令，然后一条一条执行。你点击一个按钮，CPU 处理你的点击，然后告诉屏幕显示啥。

CPU 的特点：

啥都能干，但一次干不了太多。

就像一个人，啥活儿都能接，但一次只能干一件事。你想让它同时干十件事？它得一件一件来，或者切换来切换去。

CPU 好不好，看几个指标：

核心数：几个大脑。双核就是俩大脑，四核就是四个大脑，八核就是八个大脑。大脑多，能同时干的活儿就多。

主频：大脑转多快。3.0GHz 就是每秒 30 亿次。转得快，干活就快。

缓存：大脑旁边的小本本。记着最近常用的数据，不用每次都去翻仓库。缓存大，效率就高。

举个例子：

假设你要算 1000 道数学题。

单核 CPU 就像一个人，一道一道算。8 核 CPU 就像 8 个人，每人算 125 道，同时算。

哪个快？当然是 8 个人同时算快。

但不是所有任务都能拆成 8 份。有些任务有先后顺序，必须一道一道来，这时候 8 核也不比单核快多少。

## 2、内存：工作台

内存，全叫"随机存取存储器"，你可以把它理解为计算机的工作台。

**内存干啥活儿？**

它临时存放正在使用的数据和程序。

你打开一个软件，软件的代码和数据就被加载到内存里。CPU 从内存里读取数据，处理完了再把结果写回内存。

**内存的特点：**

快，但不持久。

**快到啥程度？**

比硬盘快几十倍到上百倍。CPU 从内存读数据，眨眼就完；从硬盘读数据，得等半天（在计算机的时间尺度上）。

**不持久是啥意思？**

断电就没了。你正在写的文档没保存就断电了？内存里的数据全没了，找不回来。

**内存好不好，主要看容量：**

容量：工作台多大，用字节做衡量单位。

8GB 就是 8 吉字节，16GB 就是 16 吉字节，32GB 就是 32 吉字节。

G 是 10 的 9 次方，也就是 10 亿；B 是字节，简单可以理解为一个字母、一个汉字。

工作台大，能同时放的程序就多，开很多软件也不卡。

**你有没有遇到过这种情况：**

开了好多网页和软件，电脑变慢了，甚至提示"内存不足"？

那就是工作台满了，CPU 没地方放新东西了。这时候，操作系统会把一些暂时不用的数据挪到硬盘上（这叫"虚拟内存"），腾出工作台空间。但硬盘比内存慢多了，所以电脑就变卡了。

### **3、硬盘：仓库**

硬盘，你可以把它理解为计算机的仓库。

#### **硬盘干啥活儿？**

它永久存储数据和程序。

你下载的电影、存的照片、装的软件，都在硬盘里。关机了也不会丢。

#### **硬盘的特点：**

持久，但慢。

#### **持久是啥意思？**

断电了数据还在，放几年也没事。你 2010 年存的文档，现在还能打开，这就是硬盘的功劳。

#### **慢到啥程度？**

比内存慢几十倍到上百倍。这就是为什么计算机工作的时候，会把常用的数据从硬盘搬到内存里，让 CPU 从内存里读取，而不是每次都从硬盘读。这样干活才快。

#### **硬盘分两种：**

##### **机械硬盘（HDD）**

有转动的盘片，像老式唱片机。读写数据的时候，磁头要在盘片上移动。

优点：便宜、容量大。几百块能买好几 TB。

缺点：慢、有噪音、怕震动。

像老式仓库，东西多但取货慢。

##### **固态硬盘（SSD）**

没有转动部件，用芯片存数据，像 U 盘放大版。



优点：快、静音、抗震。

缺点：贵、容量相对小（不过现在也越来越大了）。

像新式仓库，取货快但租金高。

现在新电脑基本都是 SSD 了，速度快了很多。开机从几分钟变成几秒，打开软件也是秒开。

## 4、GPU：原本是个画画的高手，现在变成了人工智能大力士

GPU，全叫"图形处理器"。

GPU 干啥活儿？

它专门处理图像和视频。

你打游戏、看视频、做 3D 建模，都是 GPU 在干活。CPU 把图像数据给 GPU，GPU 算出每个像素点该显示啥颜色，然后输出到屏幕。

GPU 的特点：

擅长同时干很多简单活儿。

CPU 是一个大脑，一次干一件事（或者切换着干多件事）。GPU 是成千上万个"小大脑"，每个小大脑都很弱，但能同时干活。

举个例子：

你要算 100 万个简单加法题。

CPU 的做法：一个大脑，一道一道算。虽然算得快，但 100 万道也得算一会儿。

GPU 的做法：1000 个小大脑，每人算 1000 道，一起算，瞬间搞定。

这就是 GPU 的优势：并行计算。

所以，GPU 不光用来打游戏、画图，现在跑 AI 模型也靠它。AI 模型需要大量简单计算（矩阵乘法），GPU 正好擅长这个。

没有 GPU 行不行？

行，但很多事儿干不了或者干得很慢。

打游戏？没 GPU，画面卡成 PPT。

跑 AI？没 GPU，等上一天也出不来结果。

### GPU 好坏看啥？

显存：相当于 GPU 自己的工作台。显存大，能处理更大的图像、更大的模型。

核心数：小大脑的数量。核心多，算得快。

## 5、哥四个咋配合？

打个比方：

就像一个工厂：

- 硬盘是原料仓库，东西多但取货慢
- 内存是工作台，取货快但放不下太多
- CPU 是总工程师，啥都懂但一次干不了太多
- GPU 是一群流水线工人，每人干一点，加起来干得快

一个具体的例子：

你用电玩《原神》：

1. 游戏程序和资源（地图、人物、声音等）存在硬盘里
2. 你双击打开游戏，游戏被加载到内存里
3. CPU 负责处理游戏逻辑（移动、攻击、碰撞等）
4. GPU 负责渲染画面（算出每帧图像）
5. 你退出游戏，进度被保存到硬盘里

任何一个环节出问题，游戏都会受影响：

- 硬盘慢？加载时间长
- 内存小？开不了高画质
- CPU 弱？人多的时候卡
- GPU 差？画面不流畅

### 小结一下

- CPU：大脑，负责执行指令和计算，啥都能干但一次干不多

- 内存：工作台，临时放数据，快但断电就没了
- 硬盘：仓库，永久存数据，持久但慢
- GPU：画画高手，计算大力士，擅长同时干很多简单活儿，图像和 AI 靠它

## 1.3 操作系统：计算机的大管家

### 1、操作系统是干啥的？

你打开电脑，看到的是 Windows、macOS 或者 Linux 的界面。这就是操作系统。

操作系统是计算机的大管家，它管理着计算机的所有资源，让各种软件能正常运行。

没有操作系统，你的电脑就是一堆零件，啥也干不了。有了操作系统，你才能安装软件、打开文件、上网冲浪。

打个比方：

计算机硬件就像一栋大楼，有房间、水电、电梯。但没有物业，大楼就是乱的，谁也不知道咋用。

操作系统就是物业，它管理着大楼的方方面面：

- 分配房间（内存管理）
- 管理水电（硬件管理）
- 维护秩序（进程管理）
- 提供服务（系统调用）

### 2、操作系统管啥？

操作系统主要管这几件事儿：

#### 管理硬件

CPU、内存、硬盘、GPU，这些硬件资源由操作系统统一分配。哪个程序用多少内存、用多少 CPU 时间，都是操作系统说了算。

打个比方：操作系统像个调度员，安排各个程序使用硬件资源。

## 管理文件

你的文件存在硬盘上，操作系统负责组织文件、建立目录、管理权限。你能看到文件夹、能复制粘贴文件，都是操作系统的功劳。

打个比方：操作系统像个图书管理员，帮你整理和查找文件。

## 管理进程

你同时打开好几个软件，每个软件都是一个"进程"。操作系统决定哪个进程先运行、哪个后运行、哪个暂停。

打个比方：操作系统像个交通警察，指挥各个进程有序运行。

## 提供接口

操作系统给软件提供统一的接口，让软件不需要关心硬件细节。软件开发者只需要调用操作系统的功能，不需要自己去控制 CPU 和内存。

打个比方：操作系统像个翻译，软件说"我要存文件"，操作系统翻译成硬件能懂的指令。

## 3、常见的操作系统

### Windows

微软家的，全球用得最多。

优点：

- 界面友好，容易上手
- 软件丰富，啥都有
- 游戏支持好，大作基本都有 Windows 版

缺点：

- 容易中毒，安全差一点
- 系统臃肿，用久了会变慢

### macOS

苹果家的，只用在 Mac 电脑上。

优点：

- 界面漂亮，设计感强
- 系统稳定，不容易崩溃
- 设计和视频工作首选，软件优化好

缺点：

- 软件没 Windows 多
- 游戏支持差，大作很多没有 Mac 版
- 电脑贵，性价比低

## Linux

开源免费的系统，有很多发行版（Ubuntu、CentOS、Debian 等）。

优点：

- 安全稳定，不容易中毒
- 免费，不用花钱买
- 程序员喜欢，开发环境好

缺点：

- 上手难，需要学习
- 桌面软件少，日常使用不太方便

## Android

谷歌家的，用在手机和平板上。基于 Linux，开源，手机厂商都能用。

全球手机市场占有率最高。

## iOS

苹果家的，用在 iPhone 和 iPad 上。封闭系统，只能用在苹果设备上。

优点：

- 流畅、安全、软件质量高

缺点：

- 不够灵活，啥都得听苹果的

## 4、选哪个操作系统？

看你干啥：

用途	推荐系统	原因
编程、日常办公、打游戏	Windows	软件多，游戏多
编程、设计、音视频制作	macOS	软件优化好，显示好
编程、服务器、学习技术	Linux	开发环境好，免费
手机日常使用	Android 或 iOS	看个人喜好

Linux 在服务器领域是绝对主流，想学编程的话，早晚得接触 Linux。

### 虚拟机和双系统：

如果你又想用 Windows 打游戏，又想学 Linux，可以装虚拟机或者双系统。

虚拟机：在 Windows 里装一个 Linux 虚拟机，两个系统同时跑。

双系统：电脑里装两个系统，启动的时候选进哪个。

## 小结一下

- 操作系统是计算机的大管家，管理硬件、文件、进程
- 常见操作系统：Windows、macOS、Linux、Android、iOS
- 不同操作系统适合不同用途，看你干啥选啥

## 1.4 软件是咋回事儿？

### 1、软件是啥？

硬件是看得见摸得着的，软件是看不见摸不着的。

软件就是一堆代码和数据，让硬件干活的指令集合。

没有软件，计算机就是一堆废铁。有了软件，计算机才能干各种各样的事儿。

打个比方：

硬件是人的身体，软件是人的思想和技能。

没有思想，身体就是行尸走肉。有了思想，人才能干各种事儿。

## 2、软件分几种？

### 系统软件

操作系统就是系统软件，它管理整个计算机。

还有驱动程序，让操作系统能控制硬件。比如显卡驱动、声卡驱动、打印机驱动。

### 应用软件

你日常用的软件：微信、浏览器、播放器、办公软件、你单位用的各种系统.....这些都是应用软件。

应用软件跑在操作系统上，操作系统管理着硬件。

### 开发工具

写代码用的软件：VS Code、PyCharm、Git.....这些是开发工具。

开发工具也是一种应用软件，只是专门给开发者用的。

## 3、软件是咋来的？

软件是人写出来的。

程序员用编程语言写出代码，代码经过编译或解释，变成计算机能执行的指令，就成了软件。

一个软件，可能只有几百行代码，也可能有几百万行代码。比如 Windows，据说有几千万行代码。

### 举个例子：

你用的微信：

- 前端（手机 App）：可能是几百万行代码
- 后端（服务器）：可能是几千万行代码
- 各种库和框架：又是几百万行
- 加起来，可能上亿行代码

当然，大部分代码不是从零写的，而是用别人写好的库和框架组装起来的。

## 4、软件的运行

软件要运行，得经过这几步，以微信为例：

1. 安装：你下载微信安装包，双击安装，微信的文件被复制到硬盘上。
2. 加载：你点击微信图标，操作系统把微信的程序加载到内存里。
3. 执行：CPU 开始执行微信的代码，显示登录界面。
4. 交互：你扫码登录，微信显示聊天界面，你跟朋友聊天。
5. 关闭：你点击关闭，微信退出，内存里的数据被释放。

## 5、为啥有的软件装不上？

你有没有遇到过这种情况：下载了个软件，安装的时候提示"不兼容"或者"缺少依赖"？

这通常是因为：

### 操作系统不兼容

Windows 软件装不了 Mac，32 位软件跑不了 64 位系统。

就像你买了德国电器，却发现插头是欧洲标准的，插不到中国插座上。

### 缺少运行环境

有些软件需要特定的运行库，比如.NET Framework、Java Runtime、Visual C++ Runtime。

就像你买了个需要电池的玩具，但没配电池，得自己买。

### 版本不对

软件版本和系统版本不匹配。比如某软件只支持 Windows 10 及以上，你用的是 Windows 7。

### 硬件不满足

软件要求 8GB 内存，你电脑只有 4GB。软件要求独显，你电脑是集显。



## 小结一下

- 软件是一堆代码和数据，指挥硬件干活
- 软件分系统软件、应用软件、开发工具
- 软件是人用编程语言写出来的
- 软件装不上，可能是系统不兼容、缺环境、版本不对、硬件不满足

## 1.5 计算机咋跟人交流：输入输出那些事儿

### 1、人咋跟计算机说话？

你想让计算机干事儿，得跟它说话。但计算机不懂人话，咋办？

输入设备就是让你跟计算机说话的工具。

常见的输入设备：

- 键盘：输入文字、命令
- 鼠标：点击、选择、拖拽
- 触摸屏：用手直接点
- 麦克风：输入声音（语音输入）
- 摄像头：输入图像（视频通话、人脸识别）
- 扫描仪：输入纸质文档

### 2、计算机咋跟人说话？

计算机干完活儿，得告诉你结果。它咋跟你说？

输出设备就是计算机跟你说话的工具。

常见的输出设备：

- 显示器：显示图像、文字
- 扬声器：播放声音
- 打印机：打印文档
- 投影仪：投影画面
- 手机
- 头盔

### 3、标准输入输出

在编程里，有个概念叫"标准输入输出"。

标准输入：默认是键盘

标准输出：默认是显示器

标准错误输出：也是显示器，但专门输出错误信息

举个例子：

你写一个 Python 程序：

```
name = input("请输入你的名字：")           # 标准输入
print("你好，" + name)                       # 标准输出
```

运行的时候：

请输入你的名字：张三            ← 你从键盘输入

你好，张三                    ← 程序输出到显示器

### 4、终端和控制台

标准输入输出是程序与外部世界沟通的“数据流”，需要一个具体的“窗口”来承载这些交互。这个窗口就是终端或控制台。

终端：通常指一个软件程序（如 macOS 的“终端”应用、Windows 的命令提示符或 Windows Terminal），它是你输入命令和看到文字输出的那个界面。它是你与 Shell（命令解释器，如 Bash、PowerShell）直接交互的“前端”。

控制台：在日常编程中，常与“终端”混用，但更侧重于指代程序输出的显示区域。例如，你在 IDE 里运行代码，结果就显示在“运行控制台”里。

### 5、重定向

有时候，你想把输入输出换个地方。这叫"重定向"。

输出重定向：

把本来要显示在屏幕上的内容，存到文件里。

```
python hello.py > output.txt
```

本来要显示在屏幕上的内容，现在存到了 `output.txt` 文件里。

输入重定向：

不从键盘读输入，从文件读。

```
python hello.py < input.txt
```

本来要从键盘输入的内容，现在从 `input.txt` 文件里读。

## 小结一下

- 输入设备让人跟计算机说话：键盘、鼠标、触摸屏等
- 输出设备让计算机跟人说话：显示器、扬声器、打印机等
- 标准输入默认是键盘，标准输出默认是显示器
- 重定向可以改变输入输出的来源和去向

## 本章小结

这一章，咱们唠了计算机的基本概念：

计算机不是神仙，它就是个算得快的工具。能干的事儿都是能转化成计算的事儿。

计算机四大件儿：

部件	比喻	作用	特点
CPU	大脑	执行指令、计算	啥都能干，一次干不多
内存	工作台	临时存数据	快，但断电就没了
硬盘	仓库	永久存数据	持久，但慢
GPU	画画高手，AI 计算	处理图像、大规模计算	擅长并行计算

操作系统是大管家，管理硬件、文件、进程，让软件能正常运行。

软件是代码集合，是人写出来的，指挥硬件干活。

输入输出设备让人和计算机能交流。

把这些搞清楚，你就对计算机有了基本的认识。

接下来，咱们唠唠程序是咋跑起来的。

## 第二章：编程序的那点事儿

### 开场白

上一章咱们唠了计算机是啥，这一章唠唠程序是咋回事儿。

你用 AI 写代码，代码写出来了，然后呢？

代码是怎么变成能跑的程序的呢？这一章就唠这个。

搞清楚程序是咋跑起来的，你才能理解 AI 给你的代码在干啥，遇到问题才知道往哪个方向排查。

### 2.1 编程语言是个大家族

#### 1、啥是编程语言？

计算机只认识 0 和 1，但你不能每次都写 0 和 1 跟它交流，那太累了。

所以，人类发明了编程语言。编程语言就是人类和计算机之间的翻译。

你用编程语言写出代码，代码被翻译成计算机能懂的 0 和 1，计算机就能执行你的指令。

打个比方：

你和老外交流，你说中文，他说英语，互相听不懂。咋办？找个翻译。

编程语言就是翻译，你用编程语言写代码，翻译把它变成计算机能懂的 0 和 1。

## 2、编程语言的演变

编程语言的发展，经历了三代：

### 第一代：机器语言

最早的时候，程序员直接写 0 和 1。比如：

```
10110000 01100001
```

这啥意思？只有计算机知道，人看着就是天书。

你想想，写个简单的程序得敲多少个 0 和 1？还容易敲错。一个数字错了，整个程序都不对。

### 第二代：汇编语言

后来，有人发明了汇编语言，用简单的符号代替 0 和 1。比如：

```
MOV AL, 61h
```

比 0 和 1 好理解了——把 61h 这个数移动到 AL 寄存器里。但还是很难懂，而且不同 CPU 汇编语言不一样。

你在 Intel 的 CPU 上写的汇编，在 ARM 的 CPU 上可能就跑不了。

### 第三代：高级语言

再后来，高级语言出现了。比如：

```
age = 25
```

这就容易理解多了——把 25 赋值给变量 age。

高级语言让编程变得简单，普通人也能学会。你不用关心 CPU 的寄存器、内存地址这些底层细节，写代码就像写英语一样。

## 3、高级语言有啥好处？

### 易读易懂

接近自然语言，容易理解。

你看 Python 代码，基本能猜出它干啥：

```

if age >= 18:
    print("成年人")
else:
    print("未成年")

```

不需要懂计算机，就能看懂这段代码的意思。

### 移植性好

同一份代码可以在不同电脑上运行。

你用 Python 写的代码，在 Windows 上能跑，在 Mac 上也能跑，在 Linux 上还能跑。不需要为每个操作系统写不同的代码。

### 效率高

程序员写代码更快，不用关心底层细节。

以前用汇编语言写一个功能，可能要写几百行。现在用高级语言，几行就搞定。

## 4、常见的高级语言

现在有几百种编程语言，常用的也有几十种。以下是最常见的：

语言	用途	特点	难度
Python	数据分析、AI、Web、脚本	简单易学，库丰富	★☆☆☆☆
JavaScript(简称JS)	Web 前端、Node.js 后端	浏览器原生支持，无处不在	★★☆☆☆
Java	企业应用、Android 开发	跨平台，稳定，生态好	★★★★☆
C/C++	系统开发、游戏、嵌入式	性能高，但难学	★★★★★
Go	云原生、微服务	简洁高效，谷歌出品	★★☆☆☆
Rust	系统开发	安全高性能，但学习曲线陡	★★★★☆
C#	Windows 开发、游戏（Unity）	微软出品，Windows 生态好	★★☆☆☆
Swift	iOS/macOS 开发	苹果出品，开发苹果应用首选	★★☆☆☆
Kotlin	Android 开发	谷歌推荐的	★★☆☆☆

		Android 开发语言	
PHP	Web 后端	简单，但有点过时了	★ ★ ☆ ☆ ☆
Ruby	Web 后端	优雅，但用的人少了	★ ★ ☆ ☆ ☆
TypeScript	Web 前端	JavaScript 的超集，加了类型	★ ★ ★ ☆ ☆

## 5、咋选语言？

很多新手问：我想学编程，选哪个语言？

这个问题没有标准答案，因为不同语言适合干不同事儿。

**按用途选：**

用途	推荐语言
数据分析、AI、机器学习	Python
Web 前端	JavaScript
Web 后端	Python、Java、Go、Node.js(JavaScript)
企业应用	Java
手机 App（Android）	Kotlin、Java
手机 App（iOS）	Swift
跨平台 App	Flutter（Dart）、React Native（JavaScript）
游戏开发	C++（大型游戏）、C#（Unity）
系统开发、嵌入式	C、Rust
云原生、微服务	Go

**按难度选：**

最容易上手：Python

Python 语法最接近自然语言，新手学起来不痛苦。代码读起来像读英语。

中等难度：JavaScript、Java、Go

语法比 Python 复杂一些，但也不算难。JavaScript 的异步概念需要理解一下。Java 的面向对象概念需要学一学。



较难：C、C++、Rust

需要管理内存、理解指针，容易出错。学这些语言需要耐心。

**我的建议：**

如果你是纯新手，想快速入门，推荐从 Python 或 JavaScript 开始。

学完 Python 或 JavaScript，理解了编程的基本概念（变量、函数、循环等），再学其他语言就容易多了。

如果你有明确的目标（比如想搞前端），直接学那个领域的语言就行。

如果想做网站的话就先学 HTML 和 JavaScript。很多程序员不把 HTML 看作一种编程语言，因为它是做网页的。但它也是一种去分析语言，叫做“超文本标记语言”。

## 小结一下

- 编程语言是人类和计算机之间的翻译
- 从机器语言到汇编语言到高级语言，越来越容易理解
- 高级语言易读易懂、移植性好、效率高
- 常见语言：Python、JavaScript、Java、C/C++、Go、Rust 等
- 新手推荐从 Python 开始

## 2.2 代码咋变成行动的？

### 1、写完代码就能跑吗？

你写了一段代码，然后呢？直接双击能运行吗？

不一定。这取决于你用的是啥语言、啥方式。

### 2、编译型语言

有些语言需要先编译，把代码翻译成计算机能直接执行的文件。

举个例子：

C 语言的编译过程：

你写完代码，用编译器编译，生成一个.exe 文件（Windows）或可执行文件（Linux/Mac）。双击这个文件就能运行。

**编译型语言的特点：**

优点：

- 运行速度快（已经是计算机能直接执行的代码）
- 不需要运行时环境（可以直接跑）

缺点：

- 每次修改代码都要重新编译
- 不同操作系统需要重新编译

常见的编译型语言：C、C++、Go、Rust

### 3、解释型语言

有些语言不需要编译，运行的时候一行一行解释执行。

举个例子：

Python 的执行过程：

你写完代码，直接用 Python 解释器运行就行，不需要先编译。

**解释型语言的特点：**

优点：

- 不需要编译，改完代码直接运行
- 跨平台方便（只要有解释器就行）

缺点：

- 运行速度比编译型慢（边翻译边执行）

常见的解释型语言：Python、JavaScript、Ruby、PHP

### 4、中间派：Java

Java 有点特殊。它先编译成"字节码"，然后由 Java 虚拟机（JVM）解释执行。

这样既有编译型语言的优点（跨平台），又有解释型语言的灵活性。

Java 的特点：

- 写一次，到处运行（只要有 JVM）
- 比纯解释型语言快，比纯编译型语言慢
- JVM 会做优化（JIT 编译），热点代码会被编译成机器码

## 5、你的 AI 代码咋运行？

你用 AI 写的代码，运行方式取决于语言：

语言	运行方式	你需要做啥
Python	解释执行	安装 Python 解释器
JavaScript	浏览器或 Node.js 执行	有浏览器就行，或装 Node.js
Java	JVM 执行	安装 Java（JDK）
C/C++	先编译再执行	安装编译器（gcc/g++）
Go	先编译再执行	安装 Go

所以，用 AI 写代码之前，你得先搞清楚：这个代码需要啥运行环境？

AI 给你一段 Python 代码，你得有 Python 环境；给你一段 JavaScript 代码，你得有浏览器或 Node.js；给你一段 C++ 代码，你得有编译器。

## 小结一下

- 代码不能直接运行，需要翻译成计算机能懂的指令
- 编译型语言先编译成可执行文件，运行快但每次要重编译
- 解释型语言逐行解释执行，方便但运行慢
- Java 是中间派，先编译成字节码，再由 JVM 执行
- 不同语言需要不同的运行环境

## 2.3 变量、条件、循环、函数——编程就其实那么几下子

### 1、编程的核心概念

不管啥语言，编程都有几个核心概念。搞懂这些，编程的大门就推开一半了。

这些概念是：

- 变量
- 数据类型
- 条件
- 循环
- 函数

咱们一个一个唠。

### 2、变量：给数据起个名字

变量就是个"容器"，用来存数据。你给这个容器起个名字，以后用这个名字就能找到数据。

举个例子：

```
name = "张三"
age = 25
height = 1.75
is_student = True
```

这里，name、age、height、is\_student 都是变量名，存了不同的数据。

你可以把变量想象成一个个盒子，盒子上贴着标签，盒子里放着数据。

变量的命名规则：

- 名字要有意义，少用`a`、`b`、`c`这种，多用`userName`、`totalPrice`这种
- 不同语言命名规则不同，但大多不允许空格和特殊符号
- 别用语言的关键字（比如`if`、`for`、`while`）当变量名
- 建议用英文，别用中文（虽然有些语言支持。对了，别相信中文编程，真正的程序员都觉得那是个笑话。）

## 命名风格：

风格是建议，不是规则，对计算机执行没有任何影响。

你可以有你的风格，

我可以有我的风格，

阿色没有风格，也是一种风格。

风格	例子	常用于
小驼峰	userName	JavaScript、Java
大驼峰	UserName	类名
下划线	user_name	Python

## 3、数据类型：数据有啥种类

变量存的数据，有不同类型：

类型	说明	例子
整数	整数	25, -10, 0
浮点数	小数	1.75, 3.14
字符串	文本	"张三", "Hello"
布尔值	真假	True, False
列表	多个数据	[1, 2, 3]
字典	键值对	{"name": "张三"}

不同类型的数据，能干的事儿不一样：

### 【整数能加减乘除】

```
a = 10
b = 3
print(a + b)           # 13
print(a / b)           # 3.333...
```

注释：

上面的**#是注释**，就是你在程序当中做的笔记，计算机会忽略它，所以注释不参与程序的执行。

不同的语言使用不同的符号表示注释。比如：

Python 使用#；

JavaScript 使用//或/\* \*/；

C 语言、Java、Go 等很多语言都是用//或/\* \*/。

### 【字符串能拼接】

```
s1 = "你好"
s2 = "世界"
print(s1 + s2) # "你好世界"
```

但整数和字符串一般不能直接相加（有的可以，但也要尽量避免使用）

```
print("年龄：" + 25) # 报错！
print("年龄：" + str(25)) # 正确，先把 25 转成字符串
```

### 【动态类型和静态类型】

动态类型：变量类型可以变。Python、JavaScript 是动态类型。

```
x = 10      # x 是整数
x = "hello" # x 变成字符串，没问题
```

静态类型：变量类型不能变。Java、C++是静态类型。

```
int x = 10;    // x 是整数
x = "hello";   // 报错！不能把字符串赋给整数变量
```

## 4、条件：让程序有判断力

条件语句让程序能做判断，根据不同情况执行不同代码。

举个例子：

### 【判断是否成年】

```
age = 18
if age >= 18:
    print("成年人")
else:
    print("未成年")
```

程序会判断 `age >= 18` 这个条件：

- 如果是 True，执行`print("成年人")`
- 如果是 False，执行`print("未成年")`

更复杂的条件：

### 【判断是否及格】

```
score = 85
if score >= 90:      #如果...
    print("优秀")
elif score >= 80:    #否则, 如果...
    print("良好")
elif score >= 60:
    print("及格")
else:                #否则...
    print("不及格")
```

条件语句是程序"智能"的基础，没有它，程序只能傻傻地顺序执行。

组合的条件：

组合的条件能让判断更精细，常用 `and`、`or`、`not` 连接多个判断。

比如判断一个人是否可以考驾照：

### 【判断是否可以考驾照】

```
if age >= 18 and has_id_card and not is_blind:
    print("可以报名考试")
```

这里必须同时满足年龄够、有身份证且非盲人，才会进入分支。

若用 `or`，则满足任一条件即可；

`not` 用于取反，让原本为真的条件变假。

这样程序就能根据多维度信息灵活决策，像现实中的规则一样层层筛选。

掌握条件的组合写法，是写出“聪明”代码的第一步。

## 5、循环：让程序能重复干活

循环语句让程序能重复执行一段代码。

常见的循环包括 `for` 循环和 `while` 循环等方式。

`for` 循环：已知要循环几次

【打印 1 到 5】

```
for i in range(1, 6):  
    print(i)
```

输出：

```
1  
2  
3  
4  
5
```

`while` 循环：不知道要循环几次，满足条件就继续

【猜数字游戏】

```
import random  
target = random.randint(1, 100)  
guess = 0  
while guess != target:  
    guess = int(input("猜一个数字 (1-100) : "))
```



```
    if guess < target:
        print("太小了")
    elif guess > target:
        print("太大了")
print("恭喜你猜对了! ")
```

循环是程序批量处理数据的基础。没有循环，你要打印 1000 个数字就得写 1000 行代码。

循环控制：

- `break`：跳出循环
- `continue`：跳过本次，继续下次

【找到第一个能被 7 整除的数就停止】

```
for i in range(1, 100):
    if i % 7 == 0:
        print(i)
        break
```

【打印 1-10 的奇数】

```
for i in range(1, 11):
    if i % 2 == 0:
        continue # 跳过偶数
    print(i)
```

## 6、函数：打包一段功能

函数就是把一段代码打包起来，给它起个名字，以后想用这段代码，直接叫名字就行。

举个例子：

【定义一个函数，计算两个数的和】

```
def add(a, b):
    return a + b
```

【使用这个函数】

```
result = add(3, 5)
print(result) # 8
```

这里，`add` 就是一个函数，接受两个参数 `a` 和 `b`，返回它们的和。

### 函数的好处：

复用：写一遍，到处用。

你写了一个计算器，里面有加、减、乘、除四个函数。以后任何地方需要计算，直接调用这些函数就行，不用重复写。

清晰：把复杂功能拆成小函数，代码更好懂。

一个程序有 1000 行代码，如果全写在一起，看晕你。但如果你拆成 10 个函数，每个函数 100 行，就容易理解多了。

维护：改一个地方，所有调用的地方都生效。

你发现加法函数有个 `bug`，修好之后，所有调用加法的地方都自动修好了。

### 函数的组成部分：

包括函数声明关键字、函数名、形参列表、函数体、返回值。

### 以 Python 为例：

```
def add(a, b):
    res = a + b
    return res
```

### 逐部分拆解：

#### 1. 定义函数的关键字

`def`，Python 定义函数的专属关键字，必须用 `def` 开头。

#### 2. 函数名

`add`，用来调用函数，命名遵循变量规则。

#### 3. 参数列表

`(a, b)` 括号里的 `a`、`b`，接收外部传进来的数据，可以没有参数。

4. 函数体

缩进里面的代码，`res = a + b`，实现具体功能逻辑。

5. 返回值

`return res` 把结果往外传出，可以没有返回。

6. 调用演示

```
c = add(3, 5)
print(c) # 输出 8
```

小结一下

编程的五个核心概念：

概念	是啥	咋用
变量	存数据的容器	<code>`name = "张三"`</code>
数据类型	数据的种类	整数、字符串、列表等
条件	让程序做判断	<code>`if...else...`</code>
循环	让程序重复执行	<code>`for...`</code> 、 <code>`while...`</code>
函数	打包一段代码	<code>`def add(a, b): return a + b`</code>

搞懂这五个概念，编程的基础就有了。

你用 AI 写的代码，基本就是这些概念的组合。

2.4 程序是咋组织的？

1、代码文件

你写的代码，通常存在文件里。不同的语言，文件后缀名不一样：

语言	文件后缀
Python	<code>.py</code>
JavaScript	<code>.js</code>
Java	<code>.java</code>
C	<code>.c</code>

C++	.cpp
HTML	.html
CSS	.css

一个文件可以放一个程序，也可以放一个模块（一组相关的功能）。

## 2、模块和包

当程序变大，一个文件放不下，就需要拆成多个文件。

模块就是一个文件，包含一组相关的功能。

包就是多个模块放在一起的文件夹。

举个例子：

my\_project/

```
└─ main.py    # 主程序
└─ utils.py   # 工具模块
└─ models.py  # 数据模型模块
```

在 main.py 里，可以导入其他模块：

```
import utils
from models import User
from config import DATABASE_URL
```

使用导入的内容

```
result = utils.calculate(1, 2)
user = User("张三")
```

## 3、库和框架

库是别人写好的代码，你可以直接用。

框架是半成品程序，你填空就能用。

大多程序员都乐于分享自己的成果。你用他的库和框架，他会很高兴的。

但是还是要注意，有版权的一般不允许免费使用。

举个例子：

库：requests 库（发送 HTTP 请求）

```
import requests
response = requests.get("https://www.baidu.com")
print(response.text)
```

你不用自己写发送 HTTP 请求的代码，直接调用 requests 库就行。

框架：Django 框架（Web 开发）

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.home),
    path('about/', views.about),
]
```

框架给你搭好了架子，你只需要写视图函数（views）和配置路由（urlpatterns），一个网站就出来了。

常用库和框架：

语言	常用库	常用框架
Python	requests, pandas, numpy	Django, Flask, FastAPI
JavaScript	lodash, axios	React, Vue, Express
Java	Apache Commons	Spring
Go	标准库很强大	Gin, Echo

## 小结一下

- 代码存在文件里，不同语言后缀名不同
- 模块是一个文件，包是多个模块的集合
- 库是别人写好的代码，框架是半成品程序
- 用库和框架可以大大提高开发效率

## 本章小结

这一章，咱们唠了程序的基本概念：

编程语言是人类和计算机之间的翻译，从机器语言到汇编到高级语言，越来越容易理解。

不同语言适合不同用途，但也不是一定的：

- 数据分析/AI 选 Python
- 前端选 JavaScript
- 企业应用选 Java

代码运行方式：

- 编译型语言先编译后运行
- 解释型语言边解释边运行

编程核心概念：变量、数据类型、条件、循环、函数，搞懂这些，编程就入门了。

程序组织方式：文件、模块、包、库、框架，合理组织才能写出大程序。

下一章，咱们唠唠数据那些事儿。

## 第三章：数据、库和算法的那点事儿

### 开场白

计算机天天处理数据，但你真的懂数据吗？

数据是啥？数据咋存的？文件格式有啥讲究？数据库是干啥的？

这一章，咱们唠唠数据那些事儿。理解了数据，你才能理解程序在干啥，才能理解 AI 给你的代码在处理啥。

### 3.1 啥都能变成数字！

#### 1、计算机的世界观

在计算机眼里，一切都是数字。

文字？数字。

图片？数字。

声音？数字。

视频？数字。

程序？还是数字。

计算机只会处理数字，所以啥都得变成数字才能让计算机处理。

**这就像啥呢？**

就像一个只会说数字语言的怪人。你跟他说"你好"，他听不懂。你得把"你好"翻译

成数字，比如"123 456"，他才能理解。

计算机就是这样的怪人。

## 2、二进制：计算机的语言

我们平时用的是十进制，0 到 9 十个数字。

计算机用的是二进制，只有 0 和 1 两个数字。

为啥用二进制？

因为计算机是用电信号工作的，电压高就是 1，电压低就是 0。

用二进制最简单可靠。

如果用十进制，就得有 10 种电压状态，太复杂了，容易出错。

对了，现在不是有人要搞三进制计算机吗？阿色认为没什么太大意义。

十进制和二进制咋对应？

十进制	二进制	解释
0	0	0 就是 0
1	1	1 就是 1
2	10	$2 = 1 \times 2 + 0 \times 1$
3	11	$3 = 1 \times 2 + 1 \times 1$
4	100	$4 = 1 \times 4 + 0 \times 2 + 0 \times 1$
5	101	$5 = 1 \times 4 + 0 \times 2 + 1 \times 1$
6	110	$6 = 1 \times 4 + 1 \times 2 + 0 \times 1$
7	111	$7 = 1 \times 4 + 1 \times 2 + 1 \times 1$
8	1000	$8 = 1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$

不用你手算，计算机自己会转换。但你要知道，计算机内部用的是二进制。

**bit 和 byte:**

- bit（位）：二进制的最小单位，0 或 1
- byte（字节）：8 个 bit，可以表示 0-255 的整数

1 byte = 8 bits



比如：

00000000 = 0

11111111 = 255

存储单位：

单位	换算	大约
KB	1024 bytes	一段文字
MB	1024 KB	一首歌
GB	1024 MB	一部电影
TB	1024 GB	一个硬盘

### 3、文字咋变成数字？

文字怎么变成数字？用编码。

ASCII 编码：

最早的编码叫 ASCII，用数字表示英文字母和符号。

字符	ASCII 码	二进制
A	65	01000001
B	66	01000010
a	97	01100001
0	48	00110000
空格	32	00100000

ASCII 只有 128 个字符，够英文用，但中文怎么办？

Unicode 编码：

后来有了 Unicode，可以表示世界上所有的字符。

字符	Unicode	说明
中	20013	中文
文	25991	中文
A	65	英文
	128522	emoji 符号

你看到的文字，在计算机里其实是一串数字。显示的时候，计算机查表把数字转成对应的字形显示出来。

### UTF-8:

Unicode 规定了每个字符的编号，但存到文件里还需要一种方式。UTF-8 就是最常用的存储方式。

它用 1-4 个字节表示一个字符：

- 英文：1 个字节（兼容 ASCII）
- 中文：通常 3 个字节
- emoji：通常 4 个字节

如果一个英文字母对应一个汉字的话，同样字符数的中文文件比英文文件大。

但是，因为一般一个英文单词包含多个字母，而一个单词才能对应一两个汉字，所以总体上，如果表达的意思相同，往往中文文件更小。

## 4、图片咋变成数字？

图片是由很多小点组成的，每个小点叫一个像素。

每个像素有颜色，颜色用数字表示。

### RGB 模式：

用红（R）、绿（G）、蓝（B）三种颜色混合出各种颜色。每个颜色分量是 0-255。

颜色 = (R, G, B)

比如：

- 纯红色 = (255, 0, 0)
- 纯绿色 = (0, 255, 0)
- 纯蓝色 = (0, 0, 255)
- 白色 = (255, 255, 255)
- 黑色 = (0, 0, 0)

- 黄色 = (255, 255, 0)
- 紫色 = (255, 0, 255)

一张图片，就是一堆像素的集合，每个像素都是三个数字。

**举个例子：**

一张 1920×1080 的图片，有  $1920 \times 1080 = 2,073,600$  个像素。

每个像素 3 个数字（RGB），每个数字 1 个字节。

总共：  $2,073,600 \times 3 = 6,220,800$  字节  $\approx 6\text{MB}$ 。

所以原图很大，需要压缩。

**图片压缩：**

- JPEG：有损压缩，丢弃一些不重要的细节，文件小但画质下降。适合照片。
- PNG：无损压缩，画质不变但文件大。适合图标、截图。
- GIF：最多 256 色，文件小但画质差。支持动画。
- WebP：新一代格式，画质好文件小。但兼容性差点。

## 5、声音咋变成数字？

声音是波，连续的波。

但计算机只能存离散的数字，所以要把连续的波变成离散的点。这叫采样。

**采样过程：**

每隔一小段时间记录一个点的值。

**采样率：**

每秒采样多少次。

- 电话：8000 次/秒（8kHz）
- CD：44100 次/秒（44.1kHz）
- 专业音频：96000 次/秒（96kHz）或更高

采样率越高，声音越真实，文件也越大。

**位深度：**

每个采样点用多少位存储。

- 8 位：256 个级别
- 16 位：65536 个级别（CD 质量）
- 24 位：1600 万个级别（专业质量）

位深度越高，音质越好，文件也越大。

## 6、视频咋变成数字？

视频就是图片加声音。

一秒钟 24 张图片（或更多），配上声音，连起来播放，就是视频。

**帧率：**

每秒多少张图片（帧）。

- 电影：24 帧/秒
- 电视：25 或 30 帧/秒
- 游戏：60 帧/秒或更高

帧率越高，画面越流畅。

**视频为什么这么大：**

一张 1920×1080 的图片约 6MB。

一秒 30 帧就是 180MB。

一分钟就是 10.8GB。

太大了！

所以视频必须压缩。视频压缩技术非常复杂，能把文件压缩到原来的几十分之一甚至几百分之一。

## 小结一下

- 在计算机眼里，一切都是数字

- 计算机用二进制，只有 0 和 1
- 文字用编码变成数字（ASCII、Unicode、UTF-8）
- 图片由像素组成，每个像素用 RGB 数字表示颜色
- 声音通过采样变成数字
- 视频是图片加声音，数据量最大，必须压缩

## 3.2 文件格式那点秘密

### 1、文件后缀名是干啥的？

你肯定见过各种文件后缀名：.txt、.doc、.jpg、.mp4.....

后缀名是告诉操作系统和软件：这个文件是啥类型的，该用啥软件打开。

比如：

- .txt → 用记事本打开
- .doc → 用 Word 打开
- .jpg → 用图片查看器打开
- .mp4 → 用播放器打开

但后缀名只是个标识，改后缀名不会改变文件内容。

你把.jpg 改成.txt，图片还是图片，只是系统可能用文本编辑器打开它，然后看到一堆乱码。

### 2、常见文件格式

文本文件：

格式	说明	特点
.txt	纯文本	最简单，啥软件都能打开
.md	Markdown	轻量级标记语言，写文档常用
.csv	逗号分隔的数据	表格数据，Excel 能打开
.json	JavaScript 对象表示法	数据交换常用，Web 开发必备
.xml	可扩展标记语言	配置文件常用，有点啰嗦
.html	网页文件	浏览器能直接打开

.yaml	另一种数据格式	配置文件常用，比 json 简洁
-------	---------	------------------

文档文件：

格式	说明	特点
.doc/.docx	Word 文档	微软 Office，最常用
.pdf	PDF 文档	跨平台通用，不能编辑
.xls/.xlsx	Excel 表格	电子表格，数据处理
.ppt/.pptx	PowerPoint	演示文稿

图片文件：

格式	说明	特点
.jpg/.jpeg	有损压缩	照片常用，文件小
.png	无损压缩	支持透明背景，图标常用
.gif	动图	最多 256 色，支持动画
.bmp	无压缩	文件大，不常用了
.svg	矢量图	放大不失真，图标 logo 常用
.webp	新格式	画质好文件小，兼容性差点

音频文件：

格式	说明	特点
.mp3	有损压缩	最常用，音乐标配
.wav	无压缩	音质好但文件大
.flac	无损压缩	发烧友喜欢
.aac	有损压缩	比 mp3 好一点
.ogg	开源格式	游戏常用

视频文件：

格式	说明	特点
.mp4	最通用	各种设备都支持
.avi	老格式	现在用得少
.mov	苹果格式	Mac/iPhone 常用
.mkv	开源格式	支持多音轨字幕
.webm	网页格式	HTML5 视频

代码文件：

格式	说明
.py	Python 代码
.js	JavaScript 代码
.java	Java 代码
.c/.cpp	C/C++代码
.html	HTML 代码
.css	CSS 样式
.json	JSON 数据
.sql	SQL 脚本

\* 代码文件都是文本格式的，都可以用文本编辑器打开。

### 3、压缩文件

压缩文件是把多个文件打包成一个，同时压缩体积，方便传输与备份。

格式	说明	特点
.zip	最通用	各系统都支持
.rar	WinRAR 专用	压缩率高但软件收费
.7z	7-Zip 格式	压缩率最高，开源免费
.tar.gz	Linux 常用	tar 打包，gzip 压缩

### 4、文件格式选择建议

场景	推荐格式
写文档	.docx 或.md
存照片	.jpg
存截图/图标	.png
存音乐	.mp3
存视频	.mp4
存数据	.json 或.csv
存代码	看语言

## 小结一下

- 文件后缀名标识文件类型，告诉系统用啥软件打开
- 不同类型文件有不同格式，各有特点
- 文本、文档、图片、音频、视频、代码，都有各自的常用格式
- 选择格式要看场景

## 3.3 数据库：比 Excel 高级的大表格

### 1、啥是数据库？

你用过 Excel 吧？Excel 就是一个表格，存数据的。

**数据库就是更强大的“表格系统”。**

它能存更多数据、更快查询、多人同时用、保证数据安全。

### 2、为啥需要数据库？

Excel 的问题：

- 数据多了就慢（几万行就开始卡）
- 多人同时用容易冲突
- 没有权限控制（谁能看、谁能改）
- 没有事务（转账一半失败了咋办）
- 查询不方便（想查“销售额最高的前 10 个产品”得写公式）

**数据库的优点：**

**存得多**

数据库能存几百万、几千万甚至几亿条数据，查询还是很快。

**查得快**

数据库有索引，就像书的目录，查询速度快几个数量级。

**多人用**



10 个人同时查数据、改数据，不会冲突。数据库有锁机制，保证数据不会乱。

### 保证安全

突然断电，数据库能恢复到断电前的状态。这叫"事务"，要么全成功，要么全失败。

### 控制权限

谁能看、谁能改、谁能删，都能控制。财务数据只有财务能看，人事数据只有人事能看。

### 复杂查询

多表关联、聚合统计、排序分组，一条 SQL 命令搞定。

## 3、关系型数据库

最常用的是关系型数据库，数据存在"表"里，像 Excel 一样。

一张表有"列"（字段）和"行"（记录）。

比如，用户表：

id	name	age	city
1	张三	25	北京
2	李四	30	上海
3	王五	28	广州

每一列是一个字段（属性），每一行是一条记录（实体）。

多张表可以关联：

用户表：

id	name	city_id
1	张三	1
2	李四	2

城市表：

id	name
1	北京
2	上海

通过 city\_id 关联，就能查到"张三住在哪个城市"。

这就是"关系型"的含义——表之间有关系。

## 4、SQL：和数据库对话的语言

SQL（Structured Query Language）是操作关系型数据库的语言。

基本操作：增删改查

### ■ 查（SELECT）

```
SELECT * FROM users;           -- 查所有用户
SELECT * FROM users WHERE city = '北京'; -- 查北京的用户
SELECT name, age FROM users;    -- 只查名字和年龄
```

### ■ 增（INSERT）

```
INSERT INTO users (name, age, city) VALUES ('赵六', 35, '深圳');
```

### ■ 改（UPDATE）

```
UPDATE users SET age = 26 WHERE name = '张三';
```

### ■ 删（DELETE）

```
DELETE FROM users WHERE name = '王五';
```

复杂查询：

### ■ 查询用户数量

```
SELECT COUNT(*) FROM users;
```

#### ■ 查询平均年龄

```
SELECT AVG(age) FROM users;
```

#### ■ 按城市分组统计

```
SELECT city, COUNT(*) FROM users GROUP BY city;
```

#### ■ 多表关联查询

```
SELECT users.name, cities.name  
FROM users  
JOIN cities ON users.city_id = cities.id;
```

#### ■ 排序

```
SELECT * FROM users ORDER BY age DESC;
```

#### ■ 分页

从 users 表，跳过前 20 条，只取出接下来的 10 条数据

```
SELECT * FROM users LIMIT 10 OFFSET 20;
```

学会这几个命令，基本操作就够用了。

## 5、常见的关系型数据库

数据库	说明	适用场景
MySQL	最流行的开源数据库，免费	Web 应用、中小项目
PostgreSQL	功能更强大的开源数据库，免费	需要高级功能的项目
SQLite	轻量级数据库，单文件，免费	小型应用、移动应用
Oracle	商业数据库，功能强大，收费	大企业、关键系统
SQL Server	微软的数据库，收费/免费	Windows 环境、企业应用

## 6、NoSQL 数据库

除了关系型数据库，还有一些 NoSQL 数据库，适合特定场景：

### 文档数据库：

存 JSON 文档，灵活，不需要固定字段。

代表：MongoDB

```
{  
  "name": "张三",  
  "age": 25,  
  "hobbies": ["游泳", "读书"],  
  "address": {  
    "city": "北京",  
    "street": "朝阳路"  
  }  
}
```

适合：内容管理、日志、用户数据

### 键值数据库：

存键值对，简单高效。

代表：Redis

```
SET user:1:name "张三"  
GET user:1:name
```

适合：缓存、会话、计数器

### 列族数据库：

按列存储，适合大数据分析。

代表：HBase、Cassandra

适合：大数据、时间序列数据

### 图数据库：

存节点和关系，适合社交网络、推荐系统。

代表：Neo4j

适合：社交网络、知识图谱、推荐系统

## 7、JSON 和数据交换格式

JSON（JavaScript Object Notation）是一种轻量级数据交换格式，易于读写和解析。它基于键值对结构，支持字符串、数字、布尔值、数组、对象和 null 等数据类型。

举例：

```
{  
  "name": "张三",  
  "age": 30,  
  "hobbies": ["阅读", "编程"]  
}
```

常用的数据交换格式还包括 XML、YAML 等。

它们各有特点：XML 适合复杂数据结构，YAML 便于人工读写。

JSON 因其轻量和易解析，在 Web API 中应用最广泛。

## 小结一下

- 数据库是比 Excel 强大的数据存储系统，能存大量数据、快速查询
- 关系型数据库用"表"存数据，用 SQL 操作
- 常见关系型数据库：MySQL、PostgreSQL、SQLite、Oracle
- NoSQL 适合特殊场景：MongoDB（文档）、Redis（键值）、Neo4j（图）

## 3.4 数据结构和算法：程序员的内功

### 1、啥是数据结构？

数据结构是数据在计算机里的组织方式。

不同的数据结构，适合不同的场景。选对了数据结构，程序效率能提高几十倍甚至几百倍。

常见的数据结构：

### 数组（列表）

一排连续的格子，每个格子存一个数据。

[1, 2, 3, 4, 5]

- 优点：按位置访问快（直接算出位置）
- 缺点：插入删除慢（后面的都得移动）

### 链表

一串珠子，每颗珠子指向下一颗。

1 → 2 → 3 → 4 → 5

- 优点：插入删除快（改指针就行）
- 缺点：访问慢（得从头找）

### 栈

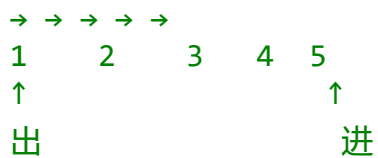
一摞盘子，只能从上面拿。



- 特点：后进先出（LIFO）
- 用途：函数调用、撤销操作

### 队列

排队买票，先来先买。



- 特点：先进先出（FIFO）

- 用途：任务队列、消息队列

## 哈希表

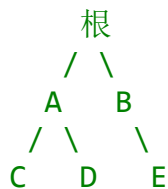
用键找值，像字典。

```
{  
  "张三": 25,  
  "李四": 30,  
  "王五": 28  
}
```

- 优点：查找快 ( $O(1)$ )
- 用途：缓存、索引

## 树

有层级关系的数据，像家谱。



- 用途：文件系统、数据库索引、HTML 结构

## 图

节点和边，像地图。



- 用途：社交网络、导航、推荐系统

## 2、啥是算法？

算法是解决问题的方法和步骤。

就像菜谱：第一步切菜，第二步放油，第三步炒.....

算法描述的是解决问题的步骤。

**常见的算法：**

**排序算法：**

把一堆乱序的数据排成有序的。

- 冒泡排序：简单但慢
- 快速排序：快，常用
- 归并排序：稳定

**查找算法：**

在一堆数据里找目标。

- 线性查找：从头找到尾，慢
- 二分查找：折半找，快（但数据得先排好序）

**图算法：**

- 最短路径：导航用
- 深度优先搜索：走迷宫
- 广度优先搜索：社交网络

### 3、算法复杂度

咋衡量算法好不好？看复杂度。

**时间复杂度：**

运行时间跟数据量的关系。

- $O(1)$ ：常数时间，跟数据量无关
- $O(\log n)$ ：对数时间，很快
- $O(n)$ ：线性时间，数据量翻倍时间翻倍
- $O(n \log n)$ ：比线性慢一点
- $O(n^2)$ ：平方时间，数据量翻倍时间翻四倍

**空间复杂度：**

占用内存跟数据量的关系。



举个例子：

在一个有序数组里找一个数：

线性查找： 从头找到尾

- 时间复杂度： $O(n)$ ，最坏情况找  $n$  次

二分查找： 折半找

- 时间复杂度： $O(\log n)$ ，100 万个数最多找 20 次

差距大了！

#### 4、你需要学吗？

如果你只是用 AI 写小程序，不需要深入学数据结构和算法。AI 会帮你选合适的。

但如果你想成为专业程序员，数据结构和算法是必修课。面试必考。

初学者建议：

- 了解基本概念就行
- 知道啥时候用啥数据结构
- 遇到性能问题，往这方面想

#### 小结一下

- 数据结构是数据的组织方式，选对了效率高
- 常见数据结构：数组、链表、栈、队列、哈希表、树、图
- 算法是解决问题的方法和步骤
- 复杂度衡量算法效率：时间复杂度和空间复杂度
- 初学者了解基本概念就行

## 本章小结

这一章，咱们唠了数据那些事儿：

一切都是数字：文字、图片、声音、视频，在计算机里都是数字。

文件格式：后缀名标识文件类型，不同类型有不同格式。

数据库：比 Excel 强大的数据存储系统，用 SQL 操作，能存大量数据快速查询。

数据结构和算法：程序员的内功，了解基本概念就行。

数据是计算机处理的对象，理解数据的概念，才能更好地理解编程。

下一章，咱们唠唠网络那些事儿。

## 第四章：网络的那点事儿

### 开场白

你天天上网，但你知道网络是咋连的吗？

你点一下链接，网页就出来了——背后发生了啥？

你发一条微信，朋友就收到了——数据是怎么过去的？

这一章，咱们唠唠网络的基本原理，让你知道点一下鼠标，背后发生了啥。

理解网络原理，你才能理解 Web 开发、API 调用、云服务这些概念。

### 4.1 两台电脑咋说话？

#### 1、网络的本质

网络的本质，就是让电脑能互相说话。

两台电脑连一根网线，就能组成最简单的网络。它们可以互相传文件、发消息。

很多电脑连在一起，就是大网络。全世界的电脑连在一起，就是互联网。

打个比方：

网络就像电话系统。每台电脑是一部电话，网线是电话线。你拨号（发请求），对方接听（响应），你们就能交流。

#### 2、网络协议：电脑的"语言"

电脑之间咋说话？得用统一的"语言"，这就是网络协议。

就像人与人交流，得说同一种语言。你说中文，我说英语，咱俩没法交流。

电脑之间也一样，得用统一的协议，才能正确传输数据。

最常见的协议是 TCP/IP，它是互联网的基础协议。

TCP/IP 不是单个协议，是一组协议的统称，成员很多，而且更新、升级、新增、废止也较频繁，主要包括：

- IP：网际协议，负责寻址和路由
- TCP：传输控制协议，负责可靠传输
- UDP：用户数据报协议，负责快速传输
- HTTP：超文本传输协议，负责网页传输
- FTP：文件传输协议，负责传文件
- SMTP：邮件传输协议，负责发邮件

这些协议各司其职，配合工作，让互联网能正常运行。

### 3、数据咋传输？

你在电脑上发个文件给另一台电脑，数据是怎么过去的？

不是一下子过去的，是被"切"成很多小包，每个包单独传，到了目的地再拼起来。

这叫分组交换。

这样做的好处：

- 某个包丢了，重传那个包就行，不用全部重传
- 多个数据可以同时传，共享网络资源
- 出错了容易定位和修复

就像你寄一本书：

不是整本寄，而是撕成很多页，每页单独装信封寄出去，对方收到后再拼成一本书。

虽然听起来麻烦，但效率更高、更可靠。

## 小结一下

- 网络的本质是让电脑能互相说话
- 网络协议是电脑交流的"语言", TCP/IP 是基础
- 数据被切成小包传输, 到目的地再拼起来

## 4.2 IP 地址：网络世界的门牌号

### 1、啥是 IP 地址？

你要寄信，得知道对方地址。电脑之间传数据，也得知道对方地址。

IP 地址就是网络世界的门牌号。

比如：192.168.1.100

这四个数字组成一个 IP 地址，每台联网的电脑都有一个 IP 地址。

### 2、IP 地址长啥样？

IPv4 地址：

4 个数字，用点分隔，每个数字 0-255。

192.168.1.100

每个数字是 8 位，即 1 个字节。8 位 2 进制数最大为 11111111，等于 10 进制的 255。

4 个数字总共 32 位，能表示的地址数量： $2^{32} \approx 43$  亿个。这就是全球所有 IP 地址的总量。

IPv6 地址：

八组十六进制数字，用冒号分隔。

2001:0db8:85a3:0000:0000:8a2e:0370:7334

总共 128 位，能表示的地址数量多到用不完。

### 3、IPv4 和 IPv6

v 就是 vision，版本的意思。v4 就是第 4 版，v6 就是第 6 版。

#### IPv4 的问题：

地址不够用了！全世界只有 40 多亿个，不够分。

你想想，全世界有多少设备要联网？电脑、手机、平板、智能手表、智能家电.....早超过 43 亿了。

#### IPv6 的解决方案：

地址数量多到几乎无穷大（ $2^{128} \approx 3.4 \times 10^{38}$  个），够用到地老天荒。

#### 现状：

IPv4 还在用，但地址不够，所以有了各种技术（NAT、动态分配、内网）来省着用。IPv6 是未来的趋势，但普及还需要时间。

### 4、公网 IP 和内网 IP

#### 公网 IP：

能在互联网上被访问的 IP 地址，全世界唯一。

就像你的家庭住址，全世界只有一个，别人可以通过这个地址找到你。

#### 内网 IP：

只在内部的局域网内有效的 IP 地址，不同局域网可以重复。

就像你小区里的门牌号，"3 号楼 502 室"，很多小区都有这个地址，但在你小区里是唯一的。

#### 常见的内网 IP 段：

- 192.168.x.x（最常见，家用路由器都用这个）
- 10.x.x.x（大公司常用）
- 172.16.x.x - 172.31.x.x

#### 你家的网络：

你的电脑、手机，用的都是内网 IP。它们通过路由器（有公网 IP）连上互联网。

这就是 NAT（网络地址转换）：多台设备共享一个公网 IP。

## 5、怎么查自己的 IP?

查内网 IP:

Windows: 打开命令行, 输入 `ipconfig`

Mac/Linux: 打开终端, 输入 `ifconfig` 或 `ip addr`

查公网 IP:

浏览器访问: <https://www.ip.cn/>

或者百度搜索"IP", 会显示你的公网 IP。

不一定准确。

## 小结一下

- IP 地址是网络世界的门牌号
- IPv4 地址快用完了, IPv6 是未来趋势
- 公网 IP 全球唯一, 内网 IP 只在局域网有效
- 家庭网络通过 NAT 让多设备共享一个公网 IP

## 4.3 域名和 DNS: 好记的名字

### 1、域名是啥?

IP 地址太难记了, 192.168.1.100 这种, 谁记得住?

于是有了域名, 比如 `www.baidu.com`, 好记多了。

域名就是 IP 地址的"别名", 方便人记忆。

### 2、域名的结构

域名从右往左看:

www.baidu.com

com → 顶级域名（.com 表示商业机构）

baidu → 二级域名（公司名）

www → 三级域名（主机名）

常见顶级域名：

顶级域名	含义
.com	商业机构
.org	非营利组织
.net	网络服务商
.edu	教育机构
.gov	政府机构
.cn	中国
.jp	日本

### 3、DNS：域名翻译官

你输入域名，电脑怎么知道对应的 IP 地址？

靠 DNS（Domain Name System），域名系统。

DNS 就像电话本，记录着域名和 IP 地址的对应关系。

查询过程：

域名查询是一个逐级扩散的过程，叫做域名解析：

用户输入域名后，本地计算机先查本地缓存，无结果就请求本地 DNS 服务器；本地 DNS 查自身缓存，没有则逐级向根域名服务器、顶级域名服务器、权威域名服务器查询，最终获取对应 IP 地址，回传给用户设备，本地计算机凭 IP 访问目标服务器。

也就是说，虽然你使用了域名，但计算机最后还是通过 IP 地址完成访问的。

DNS 缓存：

为了加快速度，电脑和 DNS 服务器都会缓存（记住）之前查过的结果。

下次再访问同一个域名，直接用缓存的 IP，不用再查了。



## 4、hosts 文件

在 DNS 之前，还有一个地方可以配置域名和 IP 的对应关系：hosts 文件。

位置：

- Windows: `C:\Windows\System32\drivers\etc\hosts`
- Mac/Linux: `/etc/hosts`

内容示例：

```
127.0.0.1    localhost
```

```
192.168.1.100 myserver
```

用途：

- 本地开发测试
- 屏蔽某些网站（比如把广告域名指向 127.0.0.1）
- 绕过 DNS 污染

## 小结一下

- 域名是 IP 地址的好记别名
- 域名从右往左：顶级域名 → 二级域名 → 三级域名
- DNS 把域名翻译成 IP 地址，像电话本
- hosts 文件可以手动配置域名和 IP 的对应关系

## 4.4 网站是咋打开的？

### 1、浏览器访问网站的过程

你在浏览器输入网址，按下回车，发生了啥？

完整的过程：

- 1) 浏览器先进行域名解析，把网址换成 IP
- 2) 建立 TCP 三次握手，和服务器连通
- 3) 发送 HTTP/HTTPS 请求（浏览网页使用的是 HTTP/HTTPS 协议）

- 4) 服务器处理请求，返回响应数据（HTML、CSS、JS、图片等）
- 5) 浏览器解析 HTML、渲染页面、加载资源、执行 JS
- 6) 浏览结束后 TCP 四次挥手断开连接

## 2、三次握手：建立连接

TCP 建立连接需要三次握手。所谓握手，就是要在正式通讯之前联络好。

为啥要三次？

确保双方都能收、能发。就像打电话：

- 1) "喂，听得到吗？"（第一次）
- 2) "听到了，你能听到我吗？"（第二次）
- 3) "听到了，你说吧。"（第三次）

## 3、HTTP 请求和响应

HTTP 请求：

浏览器发给服务器的消息，是这样的：

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0...
Accept: text/html
```

它包含的主要信息：

- 方法：GET（获取）、POST（提交）、PUT（更新）、DELETE（删除）
- 路径：/index.html
- 协议版本：HTTP/1.1
- 请求头：主机、浏览器信息、接受的内容类型等

HTTP 响应：

服务器返回给浏览器的消息：

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234
<!DOCTYPE html>
<html>
```

```
...  
</html>
```

它包含的主要信息：

- 状态码：200（成功）、404（未找到）、500（服务器错误）
- 响应头：内容类型、长度等
- 响应体：实际的 HTML 内容

服务器返回的常见状态码：

状态码	含义	说明
200	OK	成功
301	Moved Permanently	永久重定向
302	Found	临时重定向
304	Not Modified	未修改，用缓存
400	Bad Request	请求格式错误
401	Unauthorized	未授权
403	Forbidden	禁止访问
404	Not Found	未找到
500	Internal Server Error	服务器内部错误
502	Bad Gateway	网关错误
503	Service Unavailable	服务不可用

网络上的 404 这个梗就是从这里来的哦。

## 4、HTTP 和 HTTPS

HTTP（HyperText Transfer Protocol）：网页传输协议，明文传输。

HTTPS：HTTP 的加密版本，更安全。

区别：

	HTTP	HTTPS
传输方式	明文	加密
安全性	不安全	安全
端口	80	443
证书	不需要	需要 SSL 证书

现在大多数网站都用 HTTPS 了，你看看地址栏，有个小锁图标，就是 HTTPS。

### 为啥 HTTPS 安全？

HTTP 是明文传输，你发的内容可能被中间人截获看到。

HTTPS 加密传输，中间人截获了也看不懂。

### HTTPS 的加密过程：

- 1) 浏览器请求服务器的证书
- 2) 服务器返回证书（包含公钥）
- 3) 浏览器验证证书（是不是可信机构颁发的）
- 4) 浏览器生成一个随机密钥，用服务器的公钥加密，发给服务器
- 5) 服务器用自己的私钥解密，得到这个随机密钥
- 6) 之后双方用这个随机密钥加密通信

### 小结一下

- 访问网站的过程：DNS 解析→建立连接→发送请求→响应→渲染→断开
- TCP 三次握手确保双方都能收发
- HTTP 请求和响应是浏览器和服务器交流的方式
- 状态码告诉你是成功还是失败
- HTTPS 比 HTTP 安全，因为加密传输

## 4.5 常见网络概念扫盲

### 1、端口：一栋楼的房间号

IP 地址找到了服务器，但服务器上有多个程序，怎么区分？

用端口。

端口就像房间号。IP 地址是楼栋地址，端口是房间号。

IP 地址：192.168.1.100

端口号：80

完整的网络地址：192.168.1.100:80

常见端口：

端口	服务
80	HTTP（网页）
443	HTTPS（加密网页）
22	SSH（远程登录）
21	FTP（文件传输）
25	SMTP（邮件发送）
3306	MySQL 数据库
5432	PostgreSQL 数据库
6379	Redis
27017	MongoDB

端口范围：

- 0-1023：知名端口，系统占用
- 1024-49151：注册端口，应用程序用
- 49152-65535：动态端口，临时用

## 2、URL：统一资源定位器

URL（Uniform Resource Locator）是统一资源定位符的简称，用于在互联网上定位和访问资源。

每个 URL 都指向一个特定的网页、文件或服务，是网络寻址的基础。

URL 通常包含协议、域名、路径等部分，例如：

`http://www.example.com/page`

其中 http 是协议，www.example.com 是域名，/page 是路径。

通过 URL，浏览器能准确获取并显示目标内容。

## 3、防火墙：网络安全

防火墙是网络安全系统，控制哪些数据能进出。

就像小区门卫，检查每个进出的数据包，符合规则的放行，不符合的拦截。

### 防火墙规则：

- 允许某些端口的流量
- 禁止某些 IP 的访问
- 限制某些协议

### 举个例子：

服务器上运行着网站（端口 80）和数据库（端口 3306）。

### 防火墙配置：

- 允许外网访问 80 端口（用户能访问网站）
- 禁止外网访问 3306 端口（数据库不对外开放）

这样，即使数据库有漏洞，外网也攻击不到。

## 4、代理服务器：中间人

代理服务器是位于客户端和服务端之间的中间服务器。

### 正常访问：

客户端 → 服务器

### 通过代理：

客户端 → 代理服务器 → 服务器

### 代理的用途：

#### 正向代理：

客户端的代理，帮客户端访问服务器。

- 翻墙（访问被屏蔽的网站）
- 隐藏真实 IP
- 加速访问（代理服务器有缓存）

#### 反向代理：

服务器的代理，帮服务器接收请求。

- 负载均衡（把请求分发到多台服务器）

- 隐藏真实服务器 IP
- 缓存静态资源

Nginx 是最常用的反向代理服务器。

## 5、VPN：虚拟专用网络

VPN（Virtual Private Network）可以在公网上建立加密的专用通道。

正常上网：

你 → 互联网 → 网站

用 VPN：

你 → VPN 加密通道 → VPN 服务器 → 互联网 → 网站

VPN 的用途：

- 远程办公（安全访问公司内网）
- 保护隐私（流量加密，运营商看不到）
- 访问地区限制的内容

注意：在国内使用 VPN 需要遵守法律法规。

## 6、CDN：内容分发网络

CDN（Content Delivery Network）把内容缓存到各地的服务器上，用户就近访问。

没有 CDN：

用户（北京）→ 源服务器（美国）→ 延迟高

有 CDN：

用户（北京）→ CDN 节点（北京）→ 延迟低

CDN 的好处：

- 加快访问速度
- 减轻源服务器压力
- 提高可用性（一个节点挂了还有其他节点）

常见 CDN 服务商：

- 国外：Cloudflare、Akamai
- 国内：阿里云 CDN、腾讯云 CDN、百度云 CDN

## 小结一下

- 端口区分服务器上的不同程序，像房间号
- URL 是网上的统一资源定位器
- 防火墙控制网络流量，像门卫
- 代理服务器是中间人，正向代理帮客户端，反向代理帮服务器
- VPN 建立加密通道，保护隐私
- CDN 把内容分发到各地，加快访问速度

## 本章小结

这一章，咱们唠了网络的基本原理：

网络的本质是让电脑能互相说话，用协议传输数据。

IP 地址是网络世界的门牌号，IPv4 快用完了，IPv6 是未来。

域名是 IP 地址的好记别名，DNS 把域名翻译成 IP 地址。

访问网站的过程：DNS 解析→建立连接→发送请求→响应→渲染→断开。

常见概念：端口、防火墙、代理、VPN、CDN。

理解网络原理，你才能理解 Web 开发、API 调用这些概念。

下一章，咱们唠唠网页那点事儿。



# 第五章：网页和前端的那点事儿

## 开场白

你天天刷网页，但你知道网页是咋做出来的吗？

一个网页，看着花里胡哨的，有图有字有动画，点按钮还能跳转。这都是咋整的？

这一章，咱们唠唠网页的基本原理。弄明白了，你才能理解 AI 给你生成的前端代码在干啥。

## 5.1 网页的三驾马车

### 1、HTML、CSS、JavaScript

打开一个网页，你看到的是一张完整的“画”。但这幅画是怎么画出来的？

是由三样东西配合画出来的：

- HTML：骨架，决定网页有啥内容
- CSS：衣服，决定网页长啥样
- JavaScript：动作，决定网页能干啥

### 2、HTML：搭骨架

HTML（HyperText Markup Language），超文本标记语言。

听着挺高大上，其实就是用“标签”来定义网页上的各种元素。

看看一段 HTML 代码：

```

<!DOCTYPE html>
<html>
<head>
    <title>我的第一个网页</title>
</head>
<body>
    <h1>这是一个标题</h1>
    <p>这是一个段落。</p>
    
    <a href="https://www.baidu.com">这是一个链接</a>
    <ul>
        <li>列表项 1</li>
        <li>列表项 2</li>
        <li>列表项 3</li>
    </ul>
</body>
</html>

```

你打开记事本，把这段代码复制进去。

然后保存为 HTML 格式，就是文件名后缀为.html。双击这个文件就看到效果了。

这些标签都是啥意思？

标签	含义	作用
<html>	整个网页	包裹所有内容
<head>	头部	存放标题、样式等信息
<body>	主体	存放可见的内容
<h1>	一级标题	显示大号标题
<p>	段落	显示一段文字
<img>	图片	显示图片
<a>	链接	点击跳转
<ul>	无序列表	显示项目列表
<li>	列表项	列表中的每一项

HTML 就是用这些标签，把网页的骨架搭起来。

当然，不仅仅只有这些标签，还有其他更多功能的标签。

### 3、CSS：穿衣服

HTML 只有骨架，那页面得多丑啊。全是黑字白底，跟记事本似的。

CSS（Cascading Style Sheets），层叠样式表，就是给网页"穿衣服"的。

看看一段 CSS 代码：

```
h1 {  
    color: red;  
    font-size: 24px;  
    text-align: center;  
}  
p {  
    color: gray;  
    line-height: 1.5;  
    margin: 10px;  
}  
body {  
    background-color: #f0f0f0;  
    font-family: "微软雅黑", sans-serif;  
}
```

这段 CSS 在干啥？

- 标题变成红色，24 像素大，居中显示
- 段落变成灰色，行高 1.5 倍，周围有 10 像素间距
- 页面背景变成浅灰色，字体用微软雅黑

CSS 的基本语法：

```
选择器 {  
  
    属性: 值;  
  
    属性: 值;  
  
    ...  
  
}
```

- 选择器：选中你要修饰的元素（比如`h1`选中所有标题）
- 属性：你要改变啥（比如`color`改变颜色）
- 值：改成啥样（比如`red`改成红色）

## 4、JavaScript：让网页动起来

HTML 和 CSS 组合起来，页面是好看了，但还是"死"的，啥也干不了。

JavaScript 就是让网页"活"过来的语言。

看看一段 JavaScript 代码：

```
// 找到按钮元素
var button = document.getElementById("myButton");
// 给按钮添加点击事件
button.onclick = function() {
    alert("你点了我！");
};
```

这段 JavaScript 在干啥？

- 1) 找到页面上 id 为 myButton 的按钮
- 2) 给这个按钮添加一个点击事件
- 3) 当用户点击按钮时，弹出一个提示框

JavaScript 能干的事儿多了去了：

- 用户点击按钮，执行某个动作
- 用户填写表单，验证输入是否合法
- 用户滚动页面，加载更多内容
- 定时执行某些操作（比如轮播图）
- 发送网络请求，获取数据
- 操作页面元素，改变样式或内容

## 5、三者怎么配合？

一个完整的网页，HTML、CSS、JavaScript 是这样配合的：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>示例网页</title>
    <!-- CSS 写在 style 标签里，或者引入外部 CSS 文件 -->
    <style>
```

```

    body {
        font-family: sans-serif;
        background-color: #f5f5f5;
    }
    #myButton {
        padding: 10px 20px;
        background-color: #007bff;
        color: white;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
    #myButton:hover {
        background-color: #0056b3;
    }
    #message {
        margin-top: 10px;
        font-size: 18px;
        color: green;
    }
}
</style>
</head>
<body>
    <!-- HTML 定义页面结构 -->
    <h1>欢迎来到我的网页</h1>
    <p>这是一个示例网页，点击下面的按钮试试。</p>
    <button id="myButton">点我试试</button>
    <div id="message"></div>
    <!-- JavaScript 写在 script 标签里，或者引入外部 JS 文件 -->
    <script>
        // 找到按钮和消息区域
        var button = document.getElementById("myButton");
        var message = document.getElementById("message");
        // 点击按钮时执行
        button.onclick = function() {
            message.textContent = "恭喜你，按钮被点击了！";
        };
    </script>
</body>
</html>

```

这个网页的效果：

- 1) 显示一个标题和一段文字
- 2) 显示一个蓝色的按钮
- 3) 鼠标悬停按钮时，按钮颜色变深
- 4) 点击按钮，下方显示一条消息

## 小结一下

技术	作用	比喻
HTML	定义网页结构	人的骨架
CSS	美化网页样式	人的衣服
JavaScript	添加交互功能	人的动作

搞懂这三样，前端的大门就推开了。

## 5.2 前端和后端是啥关系？

### 1、啥是前端？

前端就是用户能看到的网页部分。

你在浏览器里看到的一切——文字、图片、按钮、动画——都是前端。

前端代码运行在你的浏览器里，不需要服务器参与（除了获取数据的时候）。

前端开发要做的事儿：

- 设计和实现网页界面
- 处理用户交互（点击、输入、滚动等）
- 发送请求给后端，获取或提交数据
- 把后端返回的数据展示出来

### 2、啥是后端？

后端就是服务器上的程序，用户看不见。

后端运行在服务器上，处理业务逻辑、操作数据库、返回数据给前端。

后端开发要做的事儿：

- 设计和实现业务逻辑
- 操作数据库（增删改查）
- 提供 API 接口，给前端调用
- 处理用户认证和授权
- 处理并发、安全、性能等问题

### 3、前后端怎么配合？

用一个例子来说明：

场景：用户登录

- 1) 用户输入账号密码，前端校验格式
- 2) 加密后传给后端
- 3) 后端查询数据库，校验账号密码是否匹配
- 4) 验证通过，生成会话返回前端，否则拒绝登录

前后端分工明确：

步骤	谁干	干啥
显示登录页面	前端	HTML+CSS 渲染
接收用户输入	前端	JavaScript 处理
发送请求	前端	JavaScript 发送 HTTP 请求
验证用户	后端	查数据库、验证密码
返回结果	后端	返回 JSON 数据
显示结果	前端	根据返回数据显示成功或失败

### 4、打个比方

前端 = 饭店的前厅

- 服务员接待客人
- 展示菜单
- 接受客人点菜
- 把菜端上桌

后端 = 饭店的后厨

- 厨师做菜
- 从仓库取食材
- 按订单准备
- 把做好的菜给服务员

客人 = 用户

只看到前厅（前端），看不到后厨（后端），但菜是从后厨做出来的。

实际上，前端、后端的这个叫法也就是从这里来的。

## 5、前后端分离

很久以前，前端和后端经常混在一起。网页由后端生成，前端只是显示。

现在，前后端分离是主要形态：

- 前端独立开发、独立部署
- 后端只提供 API 接口，不关心界面
- 前后端通过 HTTP 接口通信

好处：

- 前后端可以并行开发，效率高
- 前端可以用任何框架（React、Vue 等）
- 后端可以用任何语言（Python、Java、Go 等）
- 同一个后端可以服务多种前端（网页、App、小程序）

## 6、AI 编程时代的前后端

AI 编程时代，前后端又反过来走向融合了。

基于 AI 的全栈编程又火起来了。这也是阿色写这本小册子的主要原因。

你如果想全栈编程，那你就得前后端都明白。

前端界面可基于自然语言描述自动生成，后端逻辑则由 AI 根据业务需求直接架构和填充代码。

开发者的角色逐渐从编写者演变为需求定义者、审核者和优化引导者。

全栈能力被重新定义，重心转向对系统整体智能体的协调与把控。



未来，前后端的界限将变得模糊，融合在统一的智能开发流中。

## 小结一下

- 前端：用户能看到的网页部分，运行在浏览器
- 后端：服务器上的程序，用户看不见
- 前后端配合：前端发请求，后端返回数据
- 分工：前端管显示和交互，后端管业务和数据
- 分离：前后端独立开发，通过 API 通信

## 5.3 API 接口：前后端的传话筒

### 1、啥是 API？

API（Application Programming Interface），应用程序编程接口。

听着挺拗口，其实就是一种"约定"——前端发什么，后端返回什么。

打个比方：

你打电话叫外卖：

- 你说："来一份宫保鸡丁，送到 XX 小区。"
- 店里说："好的，30 分钟送到，一共 25 元。"

这就是一个接口：

- 你发的信息 = 请求
- 店里回复的信息 = 响应
- "叫外卖"这个行为 = 调用接口

### 2、接口长啥样？

一个 API 接口，通常包含这些信息：

请求信息：

- URL：接口地址，比如`/api/user/login`

- 方法：GET、POST、PUT、DELETE 等
- 参数：要传递的数据

响应信息：

- 状态码：成功还是失败
- 数据：返回的内容，通常是 JSON 格式

举个具体的例子：

接口：用户登录

请求：

```
POST /api/user/login
Content-Type: application/json
{
  "username": "zhangsan",
  "password": "123456"
}
```

响应：

```
{
  "code": 200,
  "message": "登录成功",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIs... ",
    "userId": 123,
    "username": "zhangsan"
  }
}
```

### 3、常见的请求方法

HTTP 协议定义了几种请求方法，对应不同的操作：

方法	用途	例子
GET	获取数据	获取用户信息
POST	创建数据	新增一条记录
PUT	更新数据	修改用户信息
DELETE	删除数据	删除一条记录
PATCH	部分更新	只修改某个字段

RESTful 风格:

现在流行的接口设计风格叫 RESTful, 规则是:

- 用 URL 表示资源 (名词)
- 用 HTTP 方法表示操作 (动词)

例如:

```
GET    /api/users      # 获取用户列表
GET    /api/users/1   # 获取 id 为 1 的用户
POST   /api/users     # 新增一个用户
PUT    /api/users/1   # 更新 id 为 1 的用户
DELETE /api/users/1    # 删除 id 为 1 的用户
```

## 4、接口文档

后端开发完接口, 得告诉前端怎么用。这就需要接口文档。

接口文档一般包含:

接口名称: 用户登录

接口地址: POST /api/user/login

请求参数:

- username: 用户名 (必填)
- password: 密码 (必填)

返回数据:

- code: 状态码
- message: 提示信息
- data: 返回数据
- token: 登录凭证
- userId: 用户 ID
- username: 用户名

错误码：

- 200: 成功
- 400: 参数错误
- 401: 用户名或密码错误

常用接口文档工具：

- Swagger / OpenAPI
- Apifox
- Postman
- YApi

## 5、怎么调用接口？

调用接口，是指调用一种请求方法，而一个请求方法就是一个功能。

前端调用接口的方式：

方式一：用 fetch（浏览器原生）

【调用登录接口】

```
fetch('/api/user/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    username: 'zhangsan',
    password: '123456'
  })
})
.then(response => response.json())
.then(data => {
  console.log(data);
});
```

方式二：用 axios（第三方库）

```
axios.post('/api/user/login', {
  username: 'zhangsan',
  password: '123456'
})
```

```
.then(response => {  
    console.log(response.data);  
});
```

虽然浏览器原生的 `fetch` API 已经成熟，但 `axios` 在易用性、功能完整性和兼容性上有显著优势，在实际商业项目中广泛应用。

## 小结一下

- API 接口是前后端通信的桥梁，前端请求，后端响应
- 请求包含：URL、方法、参数
- 响应包含：状态码、数据（通常是 JSON）
- RESTful 风格：用 URL 表示资源，用 HTTP 方法表示操作
- 接口文档告诉前端怎么调用接口
- 前端用 `fetch` 或 `axios` 调用接口

## 5.4 前端框架：别再手写原生代码了

### 1、为啥要框架？

你可能会想：用原生的 HTML、CSS、JavaScript 就能写网页，干嘛还要学框架？

确实能写，但项目一大，问题就来了：

**问题一：代码乱**

HTML、CSS、JavaScript 混在一起，改一处牵一发而动全身。

**问题二：重复劳动**

很多功能要反复写：按钮、表单、弹窗.....每次都从头写？

**问题三：难以维护**

别人写的代码看不懂，自己写的代码过两天也看不懂。

**问题四：效率低**

写一个简单的功能要写很多代码，累死人。

框架就是来解决这些问题的。

## 2、三大前端框架

现在前端最火的三大框架：

框架	出品	特点
React	Facebook	组件化、虚拟 DOM、生态最强
Vue	尤雨溪	渐进式、易上手、中文社区活跃
Angular	Google	大而全、企业级、学习曲线陡

### React

```
// React 组件示例
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
// 使用组件
ReactDOM.render(
  <Welcome name="张三" />,
  document.getElementById('root')
);
```

### Vue

```
<!-- Vue 组件示例 -->
<template>
  <h1>Hello, {{ name }}</h1>
</template>
<script>
export default {
  data() {
    return {
      name: '张三'
    }
  }
}
</script>
```

### 3、框架的好处

#### 组件化

把页面拆成一个个组件，每个组件可以独立开发、测试、复用。

#### 数据驱动

框架帮你处理数据和视图的同步。你只需要改数据，视图自动更新。

```
// 原生 JavaScript: 改数据还要手动改 DOM
document.getElementById('name').textContent = '李四';
// Vue: 改数据, 视图自动更新
this.name = '李四';
```

#### 生态丰富

框架有丰富的生态: UI 组件库、路由、状态管理、构建工具.....

不需要从零造轮子, 拿来用就行。

其实, 阿色作为一个老程序员, 不习惯使用框架, 还是不自觉地使用原生代码, 但这是不对的。得改啊!

### 4、选哪个框架?

#### 新手入门: Vue

Vue 上手最简单, 语法直观, 中文文档齐全, 社区活跃。

#### 生态最强: React

React 生态最丰富, 大公司用得最多, 学了找工作机会多。

#### 企业项目: Angular

Angular 大而全, 适合大型企业项目, 但学习成本高。

#### 阿色建议:

先学 Vue 入门, 理解组件化、数据驱动这些概念, 再学 React 拓宽视野。

## 小结一下

- 框架解决原生开发的痛点：代码乱、重复劳动、难维护、效率低
- 三大框架：React、Vue、Angular
- 框架好处：组件化、数据驱动、生态丰富
- 新手建议：从 Vue 开始，再学 React

## 5.5 用 AI 写前端代码

### 1、AI 能帮你写啥？

#### 写 HTML 结构

你说：帮我写一个登录页面的 HTML 结构，包括用户名、密码输入框和登录按钮。

AI 回答：

```
<div class="login-container">    <h2>用户登录</h2>

    <form class="login-form">
        <div class="form-group">
            <label for="username">用户名</label>
            <input type="text" id="username" name="username"
required>
        </div>
        <div class="form-group">
            <label for="password">密码</label>
            <input type="password" id="password"
name="password" required>
        </div>
        <button type="submit" class="login-btn">登录</button>
    </form>
</div>
```

#### 写 CSS 样式

你说：帮我给上面的登录页面写 CSS 样式，要居中显示，现代简约风格。



AI 回答:

```
.login-container {
  width: 400px;
  margin: 100px auto;
  padding: 30px;
  background: white;
  border-radius: 10px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}
.login-container h2 {
  text-align: center;
  margin-bottom: 30px;
  color: #333;
}
.form-group {
  margin-bottom: 20px;
}
.form-group label {
  display: block;
  margin-bottom: 5px;
  color: #666;
}
.form-group input {
  width: 100%;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
}
.login-btn {
  width: 100%;
  padding: 12px;
  background: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
.login-btn:hover {
  background: #0056b3;
}
```

写 JavaScript 交互

你说: 帮我给登录页面添加 JavaScript, 点击登录按钮时发送请求到/api/login。

AI 回答:

```
document.querySelector('.login-  
form').addEventListener('submit', function(e) {  
    e.preventDefault();  
    const username = document.getElementById('username').value;  
    const password = document.getElementById('password').value;  
    fetch('/api/login', {  
        method: 'POST',  
        headers: {  
            'Content-Type': 'application/json'  
        },  
        body: JSON.stringify({ username, password })  
    })  
    .then(response => response.json())  
    .then(data => {  
        if (data.code === 200) {  
            alert('登录成功');  
            // 跳转到首页  
            window.location.href = '/';  
        } else {  
            alert('登录失败: ' + data.message);  
        }  
    })  
    .catch(error => {  
        console.error('Error:', error);  
        alert('网络错误, 请重试');  
    });  
});
```

## 2、怎么让 AI 给你写好代码？

一次只说一件事

别一次性让 AI 写整个网站，一个模块一个模块来。

给清晰的描述

差：

帮我写个网页。

好：

帮我写一个产品列表页面，显示产品的图片、名称、价格，每行显示 4 个产品，响应式布局，使用 **Flexbox**。

### 提供上下文

如果要让 AI 接着之前的代码写，要把之前的代码发给 AI。

### 指定技术栈

告诉 AI 你用什么框架、什么 UI 库。例如：

帮我用 **Vue3** 和 **Element Plus** 写一个用户管理页面，包括用户列表、搜索、新增、编辑、删除功能。

## 3、小心 AI 的坑

### 坑一：样式不统一

AI 每次生成的样式可能不一样，导致页面风格不一致。

解决：让 AI 用一个统一的 CSS 框架（如 Tailwind CSS、Bootstrap），或者让 AI 写一个公共样式文件。

### 坑二：代码有 bug

AI 写的代码可能有错误，比如变量名写错、逻辑有问题。

解决：代码拿来要检查、要测试，别直接上线。

### 坑三：不兼容

AI 可能用了某些新特性，在旧浏览器上跑不了。

解决：问 AI 兼容性问题，或者指定要兼容的浏览器版本。

## 小结一下

- AI 能帮你写 HTML、CSS、JavaScript

- 要给 AI 清晰的描述和上下文
- 要指定技术栈和风格
- AI 的代码要检查、要测试

## 本章小结

这一章，咱们唠了网页那点事儿：

网页三要素：HTML 骨架、CSS 衣服、JavaScript 动作。

前后端关系：前端是门面，后端是厨房，通过 API 通信。

API 接口：前后端的传话筒，请求+响应，RESTful 风格。

前端框架：React、Vue、Angular，解决原生开发的痛点。

AI 写前端：能帮你写，但要给清晰的描述，代码要检查。

下一章，咱们唠唠后端那些事儿。

# 第六章：服务器和后端的那点事儿

## 开场白

上一章唠了前端，这一章唠唠后端。

后端，就是服务器上的那些活儿。用户看不见，但事儿都在这儿干。

你点击"登录"按钮，前端把用户名密码发过来，后端得验证、得查数据库、得返回结果。这些都是后端的活儿。

这一章，咱们唠唠后端开发的基本概念。

## 6.1 后端是干啥的？

### 1、后端的职责

后端就像是饭店的"后厨"，用户点菜（发请求），后厨做菜（处理请求），然后把菜端出来（返回数据）。

具体来说，后端要干这些活儿：

#### 处理业务逻辑

这是后端的核心。比如：

- 用户登录：验证用户名密码
- 购物下单：检查库存、计算价格、生成订单
- 发表评论：检查内容、保存数据库

#### 操作数据库

增删改查（CRUD），大部分后端工作都是在跟数据库打交道。

提供 API 接口

给前端调用的接口，返回 JSON 格式的数据。

用户认证和授权

验证用户是谁、能干啥。比如普通用户只能看，管理员能改。

处理并发

多个用户同时访问，服务器得扛得住。

保障安全

防止 SQL 注入、XSS 攻击、CSRF 攻击等各种安全问题。

2、后端处理的流程

处理流程：接收请求→解析参数→验证权限→业务处理→操作数据库→返回响应。

小结一下

- 后端处理业务逻辑、操作数据库、提供接口
- 后端还负责认证授权、处理并发、保障安全
- 处理流程：接收请求→解析参数→验证权限→业务处理→操作数据库→返回响应

6.2 后端语言哪家强？

1、常见的后端语言

后端可以用很多语言写，各有特点：

语言	特点	适用场景
Python	简单易学，框架丰富	快速开发、数据处理、AI
Java	稳定可靠，生态强大	企业级应用、大型项目
Go	高性能，并发强	微服务、云原生
Node.js（JavaScript）	前端也能写后端	全栈开发、实时应用
PHP	简单，部署方便	中小型网站

C#	微软全家桶	Windows 环境
----	-------	------------

## 2、Python：简单就是美

优点：

- 语法简单，上手快
- 框架丰富：Django、Flask、FastAPI
- 生态强大：数据处理、AI 都行

缺点：

- 性能不如编译型语言
- 并发处理相对弱一些

适合谁：

- 初学者
- 快速开发原型
- 数据驱动的应用

一个 Flask 例子：

```
from flask import Flask, jsonify, request
app = Flask(__name__)
@app.route('/api/hello', methods=['GET'])
def hello():
    name = request.args.get('name', 'World')
    return jsonify({
        'message': f'Hello, {name}!'
    })
if __name__ == '__main__':
    app.run(debug=True)
```

几十行代码就能跑起来一个 API 服务，简单吧？

## 3、Java：稳就是赢

优点：

- 稳定可靠，久经考验
- 生态强大，啥都有

- 企业首选，找工作容易

缺点：

- 语法啰嗦，代码多
- 启动慢，内存占用大

适合谁：

- 大型企业项目
- 银行、电商等关键系统
- 追求稳定的应用

一个 Spring Boot 例子：

```
@RestController
@RequestMapping("/api")
public class HelloController {
    @GetMapping("/hello")
    public Map<String, String> hello(@RequestParam(defaultValue = "World")
    String name) {
        Map<String, String> result = new HashMap<>();
        result.put("message", "Hello, " + name + "!");
        return result;
    }
}
```

比 Python 啰嗦点，但类型安全，IDE 支持好。

## 4、Go：快就是强

优点：

- 性能强，接近 C
- 并发原生支持
- 部署简单，一个二进制文件搞定

缺点：

- 生态相对年轻
- 错误处理有点麻烦

适合谁：



- 高性能服务
- 微服务架构
- 云原生应用

一个 Go 例子：

```
package main
import (
    "net/http"
    "encoding/json"
)
func hello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query().Get("name")
    if name == "" {
        name = "World"
    }
    result := map[string]string{"message": "Hello, " + name +
"!"}
    json.NewEncoder(w).Encode(result)
}
func main() {
    http.HandleFunc("/api/hello", hello)
    http.ListenAndServe(":8080", nil)
}
```

简洁高效，一看就懂。

## 5、Node.js：前端也能干后端

优点：

- JavaScript 全栈，前后端用同一门语言
- 异步 IO，高并发友好
- npm 生态丰富

缺点：

- CPU 密集型任务不太行
- 回调嵌套容易晕（虽然 async/await 解决了）

适合谁：

- 前端转全栈
- 实时应用（聊天、游戏）

- 快速原型开发

一个 Express 例子:

```
const express = require('express');
const app = express();
app.get('/api/hello', (req, res) => {
  const name = req.query.name || 'World';
  res.json({ message: Hello, ${name}! });
});
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

前端看了都亲切。

## 6、选哪个？

新手入门: Python

简单易学，快速上手，能快速看到成果。

找工作: Java

企业用得更多，岗位多，学了不愁找工作。

追求性能: Go

高性能场景，微服务架构，Go 是好选择。

前端转全栈: Node.js

前后端同一门语言，学习成本低。

我的建议:

先从 Python 开始，理解后端开发的基本概念，再根据需要学其他语言。

## 小结一下

- 后端语言各有特点: Python 简单、Java 稳定、Go 快速、Node.js 全栈
- 新手建议从 Python 开始
- 不同场景选不同语言

## 6.3 Web 框架：造轮子不如用轮子

### 1、啥是 Web 框架？

Web 框架，就是你接到一个 HTTP 请求之后帮你干后端活的工具箱。

如果没有框架，你得自己处理：

- 解析 HTTP 请求
- 路由分发
- 参数验证
- 数据库连接
- 会话管理
- 错误处理
- ...

一堆事儿，累死人。

有了框架，这些它都帮你干了，你只需要写业务逻辑。

### 2、常见的 Web 框架

语言	框架	特点
Python	Django	大而全，开箱即用
Python	Flask	小而精，灵活自由
Python	FastAPI	快速，自动生成文档
Java	Spring Boot	企业级，功能强大
Go	Gin	轻量高性能
Node.js	Express	简单灵活
Node.js	Koa	现代优雅

### 3、Django：大而全

Django 是 Python 的"重量级"框架，内置了很多功能：

- ORM（数据库操作）
- 后台管理
- 用户认证
- 表单处理

- 缓存
- ...

适合：快速开发功能齐全的网站。

一个 Django 例子：

【views.py】

```
from django.http import JsonResponse
def hello(request):
    name = request.GET.get('name', 'World')
    return JsonResponse({'message': f'Hello, {name}!'})
```

【urls.py】

```
from django.urls import path
from . import views
urlpatterns = [
    path('api/hello', views.hello),
]
```

## 4、Flask：小而精

Flask 是 Python 的"轻量级"框架，只提供最核心的功能，其他自己选。

适合：小项目、微服务、需要灵活定制。

一个 Flask 例子：

```
from flask import Flask, request, jsonify
app = Flask(__name__)
@app.route('/api/hello')
def hello():
    name = request.args.get('name', 'World')
    return jsonify({'message': f'Hello, {name}!'})
if __name__ == '__main__':
    app.run()
```

## 5、FastAPI：又快又好

FastAPI 是 Python 的新星框架，特点：

- 性能强（比 Flask 快）

- 自动生成 API 文档
- 类型提示支持好
- 异步支持好

适合：现代 API 服务。

一个 FastAPI 例子：

```
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()
class HelloResponse(BaseModel):
    message: str
@app.get('/api/hello', response_model=HelloResponse)
def hello(name: str = 'World'):
    return {'message': f'Hello, {name}!'}
```

访问/docs 还能看到自动生成的 API 文档，舒服！

## 小结一下

- Web 框架帮你处理底层细节，你只写业务逻辑
- Django 大而全，Flask 小而精，FastAPI 又快又好
- 选框架要看项目需求和个人喜好

## 6.4 数据库操作：ORM 还是 SQL？

### 1、啥是 ORM？

ORM（Object-Relational Mapping），对象关系映射。

简单说，就是用代码操作数据库，不用写 SQL。

不使用 ORM：

```
import sqlite3
conn = sqlite3.connect('test.db')
cursor = conn.cursor()
```

写 SQL 查询

```
cursor.execute("SELECT * FROM users WHERE age > 18")
users = cursor.fetchall()
conn.close()
```

使用 ORM (Django) :

```
from myapp.models import User
```

用代码查询

```
users = User.objects.filter(age__gt=18)
```

不用写 SQL，用 Python 代码就能操作数据库，直观多了。

## 2、ORM 的好处

代码更清晰

SQL 是字符串，不直观。ORM 是代码，类型安全，IDE 能提示。

跨数据库

换个数据库，代码不用改。MySQL 换成 PostgreSQL，ORM 帮你搞定。

防止 SQL 注入

ORM 会自动处理参数，不用担心 SQL 注入攻击。

提高效率

常用操作一行代码搞定，不用写重复的 SQL。

## 3、ORM 的缺点

性能问题

ORM 生成的 SQL 可能不是最优的，复杂查询性能可能下降。

学习成本

得学 ORM 的语法和概念。

灵活性差

有些复杂 SQL，ORM 写不出来，还是得写原生 SQL。

#### 4、常见的 ORM

语言	ORM
Python	Django ORM、SQLAlchemy
Java	Hibernate、MyBatis
Go	GORM
Node.js	Sequelize、TypeORM、Prisma

#### 5、建议

新手：用 ORM

ORM 简单直观，不用学 SQL 也能操作数据库。

进阶：ORM+SQL

简单操作用 ORM，复杂查询写 SQL。

高级：根据场景选

性能敏感的地方，手写 SQL 优化。

#### 小结一下

- ORM 让你用代码操作数据库，不用写 SQL
- ORM 好处：清晰、跨数据库、防注入、效率高
- ORM 缺点：性能可能差、学习成本、灵活性差
- 新手用 ORM，进阶 ORM+SQL

## 6.5 用户认证：你是谁，能干啥？

### 1、啥是用户认证？

用户认证就是解决两个问题：

1. 你是谁？（认证 Authentication）
2. 你能干啥？（授权 Authorization）

认证：验证用户身份。比如登录时验证用户名密码。

授权：给用户分配权限。比如管理员能删除，普通用户只能看。

### 2、传统的认证方式：Session

流程：

- 1) 登录成功后，服务器生成并保存 Session
- 2) 服务器将 Session ID 通过 Cookie 返回给浏览器
- 3) 浏览器后续请求自动携带此 Cookie
- 4) 服务器通过 Cookie 中的 Session ID 找到对应 Session 以验证用户身份
- 5) 验证成功后，处理请求并返回数据

Session 存在哪：

服务器内存里，或者 Redis 里。

缺点：

- 服务器要存 Session，多服务器时得共享 Session
- 扩展性差
- Cookie 容易被 CSRF 攻击

### 3、现代的认证方式：Token

流程：

- 1) 登录成功，服务器生成 Token
- 2) 服务器将 Token 返回给客户端
- 3) 客户端将 Token 存储在本地（如 localStorage）
- 4) 下次请求时，在 Header 中带上 Token



5) 服务器验证 Token，通过则返回数据

Token 的好处：

- 服务器不用存 Session，无状态
- 天然支持多服务器
- 可以前后端分离

最常见的 Token：JWT

JWT（JSON Web Token）是现在最流行的 Token 格式。

JWT 长啥样：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IuW8oOS4iSIsIm1hdCI6MTUxNjIzOTAyMn0.Sf1KxwRJSMekKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

看着乱七八糟，其实是三段：

Header.Payload.Signature

- Header：算法信息
- Payload：用户信息（用户 ID、过期时间等）
- Signature：签名，防止篡改

## 小结一下

- 用户认证解决"你是谁"和"你能干啥"
- 传统用 Session，服务器要存状态
- 现代用 Token（如 JWT），无状态，适合分布式

## 本章小结

这一章，咱们唠了后端开发那点事儿：

后端的职责：处理业务逻辑、操作数据库、提供接口、认证授权等。

后端语言：Python 简单、Java 稳定、Go 快速、Node.js 全栈。

Web 框架: Django 大而全、Flask 小而精、FastAPI 又快又好。

数据库操作: ORM 让代码更清晰, 新手推荐用 ORM。

用户认证: Session 传统, Token 现代, JWT 最流行。

# 第七章：云、部署和运维的那点事儿

## 开场白

你用 AI 写了代码，在自己电脑上跑得挺好。然后呢？

你朋友想用，得咋整？发给他源代码？那他得装环境、配依赖.....

要是几百万人想用呢？总不能一个一个发吧？

这就需要把程序放到服务器上，让大家通过网络访问。这就是"部署"。

现在流行"上云"，那"云"又是啥？

这一章，咱们唠唠云和部署的事儿。

## 7.1 "上云"不是真上天

### 1、啥是云计算？

"云"这个词，听着挺玄乎，其实就是互联网上的服务器。

你自己买服务器放家里，得操心一堆事儿：

- 买硬件（花钱）
- 找机房放（地方）
- 接网络、配电源（折腾）
- 维护、维修（烦人）
- 还得担心断电、断网、被盗.....

"上云"，就是把这些破事儿交给专业的人干。你只管租他们的服务器用就行。

打个比方：

- 以前：自己买发电机发电（自己建机房）
- 现在：用电业局的电（上云）

不用自己发电，插上插座就有电，按度数交钱。多省心！

## 2、云服务商有哪些？

国外：

厂商	说明
AWS（亚马逊）	全球老大，啥都有
Azure（微软）	企业用户多
Google Cloud	AI 能力强
Oracle Cloud	数据库强

国内：

厂商	说明
阿里云	国内老大，电商基因
腾讯云	游戏、社交生态好
华为云	政企客户多
百度云	AI 能力突出

选哪个？看你的需求。国内用户多，选国内的；有海外需求，选 AWS 或 Azure。

## 3、云服务的三种类型

云服务分三层：基础设施即服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）。

IaaS 提供虚拟化的计算资源，如服务器和存储；

PaaS 为开发者提供应用开发和部署平台；

SaaS 则直接向用户提供可用的应用程序。

这三层构成了云服务的核心架构，逐层抽象，使用户能按需获取相应服务。

你该选哪个？

- 想学运维、追求灵活：IaaS（云服务器）
- 只想写代码、不想管环境：PaaS（云平台）
- 只想用软件、不想开发：SaaS（在线软件）

## 4、云服务的计费方式

### 按量付费

用多少算多少，像交水电费。

- 优点：灵活，随时停
- 缺点：长期用下来比包年贵
- 适合：临时测试、不稳定的业务

### 包年包月

提前买一段时间的使用权，像租房子。

- 优点：长期用便宜
- 缺点：中途不用也花钱
- 适合：稳定业务

### 竞价实例

用云厂商空闲的资源，便宜但不稳定。

- 优点：便宜，可能只要一折
- 缺点：随时可能被收回
- 适合：不重要的后台任务

## 小结一下

- "云"就是租别人的电脑，省去维护硬件的麻烦
- 国内有阿里云、腾讯云、华为云，国外有 AWS、Azure
- 云服务分三层：IaaS（虚拟机）、PaaS（平台）、SaaS（软件）
- 计费方式：按量付费、包年包月、竞价实例

## 7.2 服务器：程序的安身之处

### 1、啥是服务器？

服务器，说白了就是一种专门给别人服务的计算机。

你的电脑是给你自己用的，服务器是给别人用的——别人通过网络访问它，它提供服务。

服务器和普通电脑的区别：

对比项	普通电脑	服务器
用途	个人使用	给别人提供服务
稳定性	偶尔重启没事	要 7×24 小时运行
性能	够用就行	要强悍
网络	普通网线	专线、大带宽
显示	有显示器	很多没显示器（远程管理）
环境	办公室、家里	专业机房

### 2、服务器的类型

按外形分：

类型	特点	适用场景
塔式服务器	像大号台式机	小企业、办公室
机架式服务器	扁平，能塞机柜	数据中心、大企业
刀片服务器	很薄，密度高	大规模数据中心

按用途分：

类型	特点
Web 服务器	放网站、跑 Web 应用
数据库服务器	存数据、跑数据库
文件服务器	存文件、做网盘
应用服务器	跑各种应用程序
游戏服务器	跑游戏服务端

### 3、虚拟服务器和云服务器

现在很少人自己买实体的物理服务器了，大多是租虚拟服务器或云服务器。

**虚拟服务器（VPS）：**

把一台物理服务器切成多份，每份都是独立的"小服务器"。

你租其中一份，就像有了一台自己的服务器。

**云服务器：**

比 VPS 更高级。资源可以弹性伸缩，今天要 2G 内存，明天可以升到 4G。

还能做快照备份、自动扩容、负载均衡等高级功能。

### 4、服务器的操作系统

服务器大多用 Linux，细分为 Ubuntu、CentOS、Debian 等发行版。

为啥不用 Windows？

- Linux 免费，Windows 要钱
- Linux 稳定，不容易崩
- Linux 资源占用少，省内存
- Linux 命令行强大，远程管理方便
- Linux 开源，安全漏洞修复快

当然，如果你的程序只能在 Windows 上跑，那就得用 Windows Server。

### 小结一下

- 服务器是专门给别人服务的电脑，要稳定、高性能、7×24 运行
- 现在大多租虚拟服务器或云服务器，不买物理机
- 服务器系统多用 Linux，稳定免费

## 7.3 部署：把程序送上战场

### 1、啥是部署？

你在自己电脑上写好了程序，别人怎么用？

得把程序放到服务器上，让它跑起来，这就是部署。

部署就像把你的程序送上战场，让它真正为用户服务。

### 2、部署的基本步骤

- 1) 准备服务器(购买或使用云服务器)
- 2) 安装运行环境(Python、Node.js 等)
- 3) 上传代码到服务器
- 4) 配置环境变量和数据库连接
- 5) 启动程序,测试是否能访问
- 6) 配置域名和 SSL 证书

### 3、连接服务器

买了服务器，怎么操作它？用 SSH。

SSH 是啥？

SSH (Secure Shell) 是一种安全的远程登录协议。

你在家，用 SSH 就能远程操作服务器，就像操作自己电脑一样。

怎么连？

Windows:

用 PuTTY 或 Xshell 工具。步骤如下

- 1) 下载安装 PuTTY
- 2) 打开 PuTTY
- 3) 输入服务器 IP 地址
- 4) 点 Open
- 5) 输入用户名（通常是 root）
- 6) 输入密码



Mac/Linux:

用终端。执行命令:

```
ssh root@服务器 IP
```

然后输入密码就行。

连上之后，你就可以在终端里敲命令操作服务器了。

## 4、上传代码

代码怎么传到服务器上?

方式一: Git

最推荐的方式。Git 是啥? 后面会讲, 先别管它。

你的代码托管在 GitHub 或 Gitee 上, 服务器上用 `git clone` 拉下来。

```
git clone https://github.com/yourname/yourproject.git
```

方式二: FTP/SFTP

用 FileZilla 等工具, 像操作文件管理器一样上传。

方式三: SCP

用命令行传文件。

```
scp -r 本地文件夹 root@服务器 IP:/目标路径
```

方式四: 专门的运维系统

比如宝塔面板。它就是安装在服务器上的 Web 服务, 专门用于运维服务器, 上传代码是其中最轻松的一件事。后面会讲。

## 5、启动程序

代码传上去了, 怎么运行?

答: 在服务器终端执行命令。

简单方式: 直接运行

```
python app.py
```

但这样有个问题：你关掉终端，程序就停了。

好一点的方式：后台运行

```
nohup python app.py &
```

这样关掉终端，程序还在跑。

更专业的方式：用进程管理器

【用 supervisor 或 pm2】

```
pm2 start app.py
```

进程管理器能帮你监控程序，程序挂了自动重启。

最专业的方式：用 systemd

写一个服务文件，让操作系统管理你的程序。

【/etc/systemd/system/myapp.service】

```
[Unit]
Description=My Application
After=network.target
[Service]
User=root
ExecStart=/usr/bin/python /path/to/app.py
Restart=always
[Install]
WantedBy=multi-user.target
```

然后：

```
systemctl start myapp    # 启动
systemctl enable myapp   # 开机自启
systemctl status myapp   # 查看状态
```

## 6、域名和 HTTPS

用 IP 访问太丑了，得配个域名。

## 买域名

去阿里云、腾讯云、GoDaddy 等平台买。

便宜的几十块一年，贵的几百几千都有。

## 域名解析

买了域名，得把它指向你的服务器 IP。

你要在域名服务商的域名管理后台添加解析记录：

记录类型	主机记录	记录值
A	@	服务器 IP
A	www	服务器 IP

这样，example.com 和 www.example.com 都会指向你的服务器。

## 配置 HTTPS

现在 HTTPS 是标配了，没有小绿锁，浏览器会报警告。

怎么搞 HTTPS 证书？

- 免费：Let's Encrypt，自动签发，3 个月自动更新一次，设置好就不用管了。
- 收费：阿里云、腾讯云等平台都有证书卖，有更高的信誉和更多的功能。

用 Nginx 配置证书：

```
server {  
    listen 443 ssl;  
    server_name example.com;  
    ssl_certificate /path/to/cert.pem;  
    ssl_certificate_key /path/to/key.pem;  
    location / {  
        proxy_pass http://127.0.0.1:8000;  
    }  
}
```

## 小结一下

- 部署就是把程序放到服务器上，让别人能访问
- 步骤：买服务器→连服务器→配环境→传代码→启动程序→配域名→监控维护

- 用 SSH 连接服务器，用 Git 上传代码，用 systemd 管理程序
- 域名解析指向 IP，配置 HTTPS 证书

## 7.4 Docker：把环境打包带走

### 1、环境不一致的烦恼

你在自己电脑上写好的程序，放到服务器上跑不了，报各种错。

为啥？环境不一致。

- 你的电脑是 Python 3.10，服务器是 Python 3.8
- 你的电脑装了某某库 2.0 版本，服务器是 1.0 版本
- 你的电脑是 Ubuntu，服务器是 CentOS

这些差异，会导致程序跑不起来或者行为不一致。

这就是"在我机器上能跑"的经典问题。

### 2、Docker 是啥？

Docker 解决了这个问题。

Docker 把你的程序和它需要的所有环境都打包成一个"容器"。

这个容器在任何电脑上都能跑，因为它自带了所有依赖。

打个比方：

- 普通部署：你搬家，到新家重新买家具
- Docker 部署：你开房车，到哪都是那个环境

Docker 的核心概念：

Docker 是一个容器技术，把程序+运行环境打包成一个镜像。

镜像可以在任何机器上运行，运行的镜像叫容器。

容器之间相互隔离。

### 3、Docker 的基本使用

#### 安装 Docker

去官网下载安装: <https://www.docker.com/>

Windows 和 Mac 有桌面版, Linux 用命令安装。

#### 拉取镜像

```
docker pull python:3.10
```

这会从 Docker Hub 下载 Python 3.10 的镜像。

#### 运行容器

```
docker run -it python:3.10 bash
```

这会启动一个 Python 3.10 的容器, 并进入 bash。

#### 查看容器

```
docker ps          # 查看运行中的容器
docker ps -a       # 查看所有容器
```

#### 停止容器

```
docker stop 容器 ID
```

### 4、Docker 的好处

#### 环境一致

开发、测试、生产环境完全一样, 再也不会有"在我机器上能跑"的问题。

#### 快速部署

一个命令就能启动, 不用每次都装环境。

#### 隔离运行

多个程序互不干扰, 你用 Python 3.8, 我用 Python 3.10, 各跑各的容器。

## 方便迁移

容器可以轻松迁移到其他服务器，打包带走就行。

## 版本管理

镜像可以打标签，版本管理很方便。出问题可以快速回滚。

## 小结一下

- Docker 解决环境不一致问题，把程序和环境打包成容器
- 核心概念：镜像（模板）、容器（运行的实例）、仓库（存镜像的地方）
- Dockerfile 是构建镜像的"配方"
- Docker 让部署更简单、更可靠

## 7.5 运维：让程序稳定跑下去

### 1、运维是干啥的？

程序部署好了，不是就完事了。你得保证它一直能跑，别动不动就崩了。

运维，就是维护程序的稳定运行。

包括：监控程序状态、处理故障、备份数据、更新版本、优化性能、安全防护……

### 2、运维的两种方式

#### 方式一：命令行（硬核派）

用 SSH 连上服务器，敲各种 Linux 命令。

- 查看日志：tail -f /var/log/nginx/access.log
- 查看进程：ps aux | grep python
- 查看磁盘：df -h
- 查看内存：free -m
- 查看网络：netstat -tuln

这种方式很灵活，但门槛高，容易敲错命令把服务器搞崩。

## 方式二：图形化面板（懒人派）

用 Web 界面操作服务器，像操作 Windows 一样简单。

最流行的就是宝塔面板。

## 3、运维的核心任务

### 监控

要随时知道程序跑得怎么样。

- 监控 CPU、内存、磁盘、网络：看资源够不够用
- 监控服务状态：看程序是不是在跑
- 监控日志：看有没有报错
- 监控访问量：看业务是否正常

宝塔面板有监控功能，也可以用专业的监控工具（如 Prometheus+Grafana）。

### 备份

数据是无价的，一定要定期备份。

- 备份代码：Git 本身就有版本管理，但最好再打个包存到别处
- 备份数据库：每天自动导出 SQL 文件，存到对象存储或另一台服务器
- 备份文件：用户上传的图片、文件等

宝塔可以设置计划任务，定时备份网站和数据库。

### 更新

程序要定期更新，修复漏洞、增加功能。

- 代码更新：Git pull 拉取最新代码
- 依赖更新：pip install -r requirements.txt --upgrade
- 系统更新：yum update 或 apt update

更新前一定要备份，测试没问题再上线。

### 安全

服务器安全很重要，别被黑客搞了。

- 改 SSH 默认端口（22 改成别的）

- 用密钥登录，不用密码
- 防火墙只开放必要端口（80、443、SSH 端口）
- 定期更新系统补丁
- 安装安全防护软件（如 fail2ban）
- 宝塔面板改默认端口（8888 改成别的），强密码

## 4、运维工具推荐

- 宝塔面板：Web 管理面板，新手友好
- Prometheus+Grafana：专业监控，画图好看
- Jenkins：自动化部署，代码提交自动构建发布
- Ansible：批量管理多台服务器
- ELK/EFK：日志收集和分析

## 5、宝塔面板：运维神器

宝塔面板是啥？

一个安装在服务器上的 Web 管理面板，让你不用敲命令就能管理服务器。

就像给你的服务器装了个“桌面”，点点鼠标就能搞定大部分运维工作。

为啥推荐宝塔？

- 免费版功能足够个人和小团队用
- 图形化操作，不用记命令
- 一键安装环境（Nginx、MySQL、PHP、Python 等）
- 一键部署网站、FTP、数据库
- 可视化文件管理，像操作网盘一样
- 监控 CPU、内存、磁盘、网络
- 安全防护、防火墙配置
- 计划任务、备份管理

怎么安装宝塔？

连上服务器（用 SSH），执行一行命令就行（CentOS 示例）：

```
yum install -y wget && wget -O install.sh  
http://download.bt.cn/install/install_6.0.sh && sh install.sh
```

安装完会显示面板地址、用户名和密码。要立即修改密码，并记住。



## 怎么用宝塔？

- 1)浏览器打开面板地址（如 `http://服务器的 IP:8888`）
- 2)输入用户名密码登录
- 3)首次登录会推荐安装环境套件（LNMP 或 LAMP），勾选需要的软件，点一键安装
- 4)等待安装完成，就可以在面板里管理网站、数据库、文件了

## 宝塔的核心功能

- 网站管理：添加网站、配置域名、SSL 证书（一键申请 Let's Encrypt 免费证书）、伪静态、反向代理
- 数据库管理：创建数据库、用户、导入导出 SQL 文件
- 文件管理：上传下载文件、在线编辑代码、修改权限
- 软件商店：一键安装 WordPress、Discuz、Node.js、Redis 等应用
- 安全：防火墙、SSH 端口修改、IP 黑名单、木马扫描
- 监控：实时查看 CPU、内存、磁盘、网络流量
- 计划任务：定时备份网站、数据库、执行脚本
- 日志：查看网站访问日志、错误日志

## 宝塔的优缺点

### 优点：

- 极大降低运维门槛，新手也能轻松管理服务器
- 功能全面，覆盖了大部分日常运维需求
- 可视化操作，效率高，不容易出错

### 缺点：

- 占用服务器资源（内存、CPU）
- 有安全风险（Web 面板容易被攻击，要改默认端口、强密码）
- 太依赖面板，不利于学习 Linux 命令

## 小结一下

- 运维就是让程序稳定跑下去，包括监控、备份、更新、安全

- 宝塔面板是运维神器，图形化操作，新手也能轻松管理服务器
- 核心任务：监控资源、定期备份、安全防护、版本更新
- 工具推荐：宝塔（日常管理）、Prometheus（监控）、Jenkins（自动化）

## 本章小结

这一章，咱们唠了云和部署的事儿：

云计算就是租别人的电脑，不用自己维护硬件。云服务商有阿里云、腾讯云、AWS 等。

服务器是专门给别人服务的电脑，要多稳定、高性能。现在大多租云服务器。

部署是把程序放到服务器上，让别人能访问。步骤：买服务器→配环境→传代码→启动→配域名→监控。

Docker 把程序和环境打包，解决了"在我机器上能跑"的问题，让部署更简单。

运维是确保程序稳定运行的活儿。宝塔面板就是个 Web 管理界面，让你能用鼠标点点就装软件、管网站、看监控，大大降低了服务器运维的门槛。

理解了云、部署和运维，你才能把 AI 写的代码真正用起来，让别人也能访问。

下一章，咱们唠唠 AI 和大语言模型的那点事儿。

# 第八章：AI 和大语言模型的那点事儿

## 开场白

前面几章，咱们聊了计算机、编程、数据、网络、前端、后端、部署这些硬核话题。

现在，终于轮到 AI 了。

你可能已经在用 AI 写代码了——用 ChatGPT、Claude、文心一言什么的。你发现 AI 能写代码、能解释代码、能找 bug，挺好用的。

但你有没有想过这些问题：

AI 到底是咋工作的？它真的“理解”代码吗？它为啥有时候特别聪明，有时候又特别傻？大语言模型和之前的 AI 有啥区别？

这一章，咱们就唠唠 AI 和大语言模型的那点事儿。不用公式，不扯玄学，就当听故事。

## 8.1 AI 是个啥？跟传统程序有啥不同？

### 1、啥是人工智能？

人工智能，Artificial Intelligence，简称 AI。听着高大上，说白了就是：让机器干点需要人脑才能干的事儿。

比如认字、识图、说话、翻译、下棋、开车、写代码……这些事儿，以前只有人能干。现在机器也能干了，这就是 AI。

但“智能”这词儿容易让人想多了。很多人（包括阿色）认为 AI 是有意识的、能思

考的、有感情的——但这可能是想多了。你让他编程的时候，最好不要这样想。

阿色打个比方：

你养了一条狗，它会坐下、握手、打滚。你觉得这条狗“通人性”。但你真的觉得它有哲学思考能力吗？没有。它会变程序吗？不会。

现在的 AI 差不多也是这样——它表现得像是会思考，但本质上是个超级复杂的“找规律机器”。不管 AI 到底有没有思想，有没有感情，你编程的时候最好把它当成一个机器看待。

## 2、图灵测试：你怎么知道对面是人是机器？

1950 年，英国数学家艾伦·图灵提出了一个测试：一个人通过键盘和对面聊天，如果这人分辨不出对面是人是机器，那这台机器就通过了图灵测试。

这其实是个很有哲学意味的问题：啥叫“能思考”？如果表现得像在思考，算不算会思考？

到了 2025、2026 年，你随便打开一个 AI 聊天，基本已经分不清对面是人是机器了。尤其是那些大语言模型，聊天水平已经远超当时的想象。

但图灵测试有个问题：它测试的是“能不能骗过你”，不是“有没有真正的理解”。

这话题有点深了，咱们点到为止。

## 3、AI 和传统程序的区别

传统程序：程序员写好规则，计算机按规则执行。输入 A，通过规则算出来 B。一切都在程序员的预料之中。

AI 程序：程序员不写具体规则，而是让程序从数据中“学”会规则。输入一堆例子，程序自己总结出规律。

打个比方：

传统程序：你给孩子写一本详细的菜谱，每一步都写清楚——“把鸡蛋打在碗里，搅拌 30 下……”孩子照着做。

AI 程序：你给孩子看 1000 个做番茄炒蛋的视频，孩子自己琢磨出“哦，大概是这样做的”，然后自己发挥。

能看出来区别了吧？

传统程序确定性高——同样的输入，永远同样的输出。AI 有“创意”——同样的提示词，每次给你的答案可能都不一样。

传统程序好调试——出错了你能追着逻辑链找 bug。AI 是黑盒子——它给你一个答案，但说不清是咋得出的。

## 4、机器学习和深度学习

机器学习（Machine Learning, ML）是 AI 的一个分支。核心思想：不写规则，让程序从数据中学。

深度学习（Deep Learning）是机器学习的一个分支。它用“神经网络”来学习，网络层数多、参数大，所以叫“深”。

神经网络是啥？说白了就是模仿人脑的神经元结构。人脑有几百亿个神经元，互相连接。人工神经网络就是模拟这个结构——用数字化的“神经元”一层层传递信息。

大语言模型（Large Language Model, LLM），就是基于深度学习的、专门处理语言的大模型。它是深度学习在自然语言处理领域的最新成果。

它们的关系是这样的：

AI（最大）→ 机器学习（子集）→ 深度学习（子集）→ 大语言模型（子集）

你现在用的 ChatGPT、Claude、通义千问、文心一言，都属于大语言模型。

## 小结一下

- AI 就是让机器干脑力活儿
- 图灵测试测的是“像不像人”，不测“有没有意识”
- 传统程序靠规则，AI 程序靠数据学习
- 机器学习→深度学习→大语言模型，是一个比一个具体的概念

## 8.2 大语言模型是咋工作的？

### 1、大语言模型的“大”是啥意思？

大语言模型，关键是“大”和“语言”。

先说“大”：以前的 AI 模型参数少，几百几千万就算大了。现在的 LLM，参数动辄几百亿甚至上万亿。GPT-4 据说有 1.8 万亿参数。

啥是参数？你可以理解为模型的“记忆细胞”或者“旋钮”。每个参数就是一个可以调节的数字，模型就是靠几亿几百亿个这样的数字来“记住”它学到的东西。

参数越多，模型能记住的规律就越复杂、越精细。但参数多也不全是好处——训练成本高、运行速度慢、容易“死记硬背”。

再说“语言”：这些模型是专门处理语言的。它们读海量的文本，学习语言的规律、知识、逻辑、甚至某种程度的“常识”。

## 2、预测下一个词的核心玩法

大语言模型的核心玩法，简单得让你不敢相信：

给它一段文字，让它预测下一个最可能出现的词。

就这么简单。

比如你输入：“今天天气真”，模型会预测下一个词大概率是“好”或者“不错”或者“热”。

然后模型把预测出的词接上去，继续预测下一个：“今天天气真好” → 预测“啊” → “今天天气真好啊” → 预测“，” → “今天天气真好啊，” → 预测“我” → .....

整个对话过程，就是这样一个词一个词“挤”出来的。看起来像在思考，其实是在做高速的“猜词游戏”。

你可能会说：这不就是个高级输入法吗？

对，也没错。但这个“输入法”看了整个互联网的数据（几万亿个词），所以它“猜”得极其准，看起来就像真的懂了一样。

当这个“猜词游戏”做到极致，猜出来的内容就不再是简单的词语接龙，而是一段段有逻辑、有知识、甚至看起来有思想的文字。

## 3、训练和推理

大语言模型有两个阶段：训练和推理。

训练阶段：

拿海量数据（互联网上的各种文字——网页、书籍、论文、代码、对话……）给模型看，让模型不断练习“猜词”。

猜错了，模型就调整自己的参数（那几百亿个“旋钮”）让下次猜得更准。

这个过程需要巨大的算力——成千上万块 GPU 同时跑几个月，电费都要几千万。

### 推理阶段：

训练好的模型，就可以用来回答问题了。你把提示词发给模型，模型运行一次，输出答案。

推理不需要那么大的算力，一块 GPU 甚至 CPU 就能跑（当然速度慢些）。

你每次用 ChatGPT 或者 Claude 聊天，都是在进行推理——模型根据你的输入，一个词一个词地生成回答。

## 4、Token 是啥玩意儿？

你可能会注意到，AI 聊天界面下面有时会显示“已使用 X 个 tokens”或者提示“最大 token 数限制”。

Token（词元）是大语言模型处理文本的基本单位。它不是“字”也不是“词”，而是一个更灵活的分片。

比如：“今天天气真好”可能被分成[“今天”，“天气”，“真好”]三个 token，也可能分成[“今天”，“天气”，“真”，“好”]四个 token。

英文类似：“I love programming”可能是[“I”，“love”，“pro”，“gramming”]这样分的。

不同模型的分词方式不一样，但大致遵循“常用词完整保留，生僻词拆成小块”的原则。

一般来说：

- 1 个汉字  $\approx$  1-2 个 token
- 1 个英文单词  $\approx$  1-2 个 token
- 1000 个 token  $\approx$  750 个英文词，或约 500 个汉字

了解 token 的概念很重要，因为：

AI 有上下文窗口限制（比如 8K、32K、128K token），超出就“记不住”前面的内容了。

计费也是按 token 算的，输入输出都花钱。所以长短问题直接影响钱包。

## 5、幻觉：AI 为啥会胡说八道？

用 AI 多了你就会发现：AI 有时候会特别自信地给出完全错误的信息。

你问它“2025 年 NBA 总冠军是谁？”它可能给你一个煞有其事的回答——球员名字、比分、MVP 都有——但全是编的。

这就是“幻觉”（Hallucination）。

为啥会这样？因为模型本质上就是在做“猜词游戏”。它学到的规律是：“当被问到 XX 问题时，通常的回答格式是 XXX。”所以它会按照这个格式生成内容。

但它并不“知道”事实是什么。它没有知识库，没有数据库，只有从训练数据中学到的统计规律。

所以：

- AI 说的不一定对——永远要核实
- AI 越自信越要小心——它没有“不确定”的概念
- 代码也要测试——AI 写的代码可能看起来对，但运行起来报错

## 小结一下

- 大语言模型的“大”指参数多，“语言”指处理文本
- 核心玩法：预测下一个词，一个词一个词地生成回答
- 训练需要海量数据和算力，推理就是日常使用
- Token 是 AI 处理文本的基本单位，影响上下文长度和费用
- 幻觉是 AI 会自信地胡说八道，要学会核实

## 8.3 常见的大语言模型

### 1、国外的选手

#### ChatGPT / GPT-4 / GPT-4o（OpenAI）

AI 界的当红炸子鸡。2022 年底 ChatGPT 发布，直接引爆了这一波 AI 热潮。GPT-



4 和 GPT-4o 能力很强，支持文本、图片、音频多模态。

优点：能力强、生态好、插件多。缺点：贵、国内访问不方便。

### **Claude (Anthropic)**

由前 OpenAI 员工创立，主打“安全”和“可靠”。Claude 在编程方面口碑很好，上下文窗口巨大（200K tokens），写长文、分析大项目很爽。

优点：编程能力强、上下文长、安全可靠。缺点：知名度不如 ChatGPT。

### **Gemini (Google)**

Google 的 AI 模型，跟 Google 生态深度整合。多模态能力强，能处理视频、图片、文字。

优点：跟 Google 产品集成好。缺点：更新有点乱，版本多。

还有 Llama（Meta 开源）、Mistral（法国开源）等，这里就不一一展开了。

## **2、国内的选手**

### **通义千问（阿里）**

阿里出品，综合能力不错，有开源版本。跟阿里云、钉钉等产品集成。

### **文心一言（百度）**

百度出品，基于文心大模型。中文理解能力强，跟百度搜索、文库等产品打通。

### **智谱清言（智谱 AI / GLM）**

清华系创业公司，GLM 模型。在学术界和工业界口碑都不错，也有开源版本。

### **豆包（字节跳动）**

字节跳动出品，集成在抖音、飞书等产品中。用户体验做得不错。

### **DeepSeek（深度求索）**

国货之光！主打极致推理、代码编写与深度逻辑思考能力，专业领域实力突出，开源生态完善，长文本处理、数理解题、工程实操表现亮眼，深受技术开发者与专业人群青睐。

还有讯飞星火、360 智脑、腾讯混元、百川智能等。国内的大模型百花齐放，各有

特色。

### 3、选哪个好？

没有“最好”的模型，只有“最适合”你的。

写代码：Claude 和 ChatGPT 都不错，很多程序员觉得 Claude 更靠谱。

日常问答：各家差异不大，挑个顺手的。

中文支持：国内模型在中文上通常更好一些。

需要联网功能：ChatGPT 和国内模型都有联网选项。

要性价比：试试国产模型，很多有免费额度或者很便宜。

阿色的建议：别纠结，先拿一个用起来。用熟了再换，反正切换成本不高。

#### 【注意】关于 Claude：

说起 Claude，大家容易混淆两个意思。

Anthropic 这个公司有两个产品，一是 Claude 大语言模型；二是 Claude Code 编程环境，它是个智能体。

Claude 大语言模型因为受控制我们用不了。

Claude Code 作为一个智能体和编程环境，我们是可以使用的，可以用国内的大语言模型支撑它。

阿色自己常用的就是 Claude Code + (Kimi Code 或 DeepSeek)，编程及修改这本书都使用了它。效果感觉非常好。

现在你看到的这个第八章，实际上是阿色最后临时加进来的。

我要求 Claude Code 插入第八章，并遵守原书的格式。

它照原样做到了，完成的非常好。能达到这样的效果主要有两个原因：一是 Claude Code 这个智能体确实非常棒；二是 DeepSeek 作为支撑它的大模型表现也非常出色。

## 小结一下

- 国外主流：ChatGPT、Claude、Gemini
- 国内主流：DeepSeek、通义千问、文心一言、智谱清言、豆包
- 写代码推荐 DeepSeek、Kimi Code、Claude、ChatGPT
- 没有最好，只有最适合，先用起来再调整

## 8.4 跟 AI 聊天：提示词那点学问

### 1、啥是提示词？

提示词（Prompt），就是你对 AI 说的话。你告诉 AI 你要啥，AI 根据这个生成回答。

别小看提示词。同一个 AI，提示词写得好，答案质量天差地别。

这就好比你跟一个超级聪明但完全没主见的助手说话——你说得越清楚，他干得越好；你含含糊糊，他就瞎猜。

### 2、写提示词的基本技巧

#### 第一：说清楚你要啥。

不好的问法：“帮我写个程序。”

好的问法：“帮我用 Python 写一个计算器程序，支持加减乘除，命令行交互式的，用户输入两个数和运算符，输出结果。”

#### 第二：给角色设定。

让 AI 扮演特定角色，回答会更专业。“你是一个资深 Python 开发者，请帮我 review 以下代码.....”

#### 第三：给示例。

你想让 AI 输出什么格式，给它一个例子。它就会照着你的格式来。

#### 第四：分步骤说。

复杂任务别一口气全扔给 AI。先让它做第一步，审核没问题了再做第二步。

## 第五：要求 AI 一步步思考。

遇到逻辑问题，在提示词里加上“请一步步思考”或者 Let's think step by step，AI 的回答质量通常会提升不少。

## 3、进阶技巧

**使用系统提示词（System Prompt）：**有些 AI（如 Claude、ChatGPT）支持设置系统提示词，相当于给 AI 一个“背景设定”，让它从头到尾都按这个风格回答。

**对话上下文管理：**AI 能记住当前对话的内容，但记不住别的对话。所以遇到复杂问题，保持在一个对话里，别开新对话。

**文件上传：**很多 AI 现在支持上传图片、PDF、代码文件等。直接上传文件让它分析，比复制粘贴方便。

**工具使用：**AI 可以调用插件、搜索网页、运行代码等。善用这些工具能大幅提升 AI 的能力。

## 4、别把 AI 当神仙

虽然 AI 很强大，但它不是万能的。你得清楚它的底线：

AI 不理解代码，它只是见过很多代码。“像”正确不等于“是”正确的。

AI 会出错，而且有时候错得离谱。永远保持警惕。

AI 没有隐私概念。不要给 AI 发送密码、密钥、身份证号等敏感信息。

AI 有知识截止日期。它不知道它训练完之后发生的事情。

把 AI 当成一个“聪明但不太靠谱的助手”，而不是“万能的神”。这个心态很重要。

## 小结一下

- 提示词就是你对 AI 说的话，说得越清楚，AI 干得越好
- 基本技巧：说清楚需求、给角色设定、给示例、分步骤、要求一步步思考
- 进阶：系统提示词、上下文管理、文件上传、工具使用
- AI 不是神仙，会出错、没隐私、有知识截止期

## 8.5 AI 编程的正确姿势

### 1、AI 能帮咱干啥？

AI 编程现在已经是很多程序员的日常了。它能做的事情真不少：

**写代码：**从零写一个功能、写测试、写脚本，AI 都能干。

**解释代码：**看不懂的代码扔给 AI，让它解释。比自己查文档快。

**找 bug：**把报错信息贴给 AI，它能帮你定位问题。

**重构代码：**让 AI 帮你优化代码结构、提高可读性。

**写文档：**自动生成注释、README、API 文档。

**学习新东西：**想学新技术、新语言，让 AI 当你的私人老师。

**更重要的是：**AI 编程极大地降低了编程的门槛。不需要背语法、不需要记 API、不需要深究细节，你只需要知道你要啥，AI 帮你实现。

### 2、AI 不能干啥？

了解 AI 的边界，才能用好 AI：

AI 不能保证正确性。它写的代码看起来对，但不一定真的对。测试和审查还是得你来。

AI 不能理解业务需求。你告诉它做个电商网站，它真给你写一个。但业务逻辑对吗？符合你的商业模式吗？这些得你来判断。

AI 不能做架构决策。用啥数据库、怎么拆分服务、要不要微服务.....这些得你来定。AI 的建议可以听，但不能照搬。

AI 不能处理你没有描述清楚的需求。你给了模糊的提示词，它就给你模糊的答案——看起来对，但总差那么点意思。

AI 不擅长版本升级、依赖冲突、环境配置这些“脏活累活”。这些还是得靠人。

### 3、怎么用好 AI 编程？

阿色总结了几条经验：

### **第一条：先想清楚再让 AI 写。**

别一上来就让 AI 写代码。你得先想清楚你要啥——功能需求、输入输出、边界条件。想清楚了，AI 写的代码才对路。

### **第二条：分解任务，逐个击破。**

别让 AI 一次性写一个几千行的程序。拆成小任务：先写用户登录，再写商品展示，再写购物车……**每一步都验证。**

### **第三条：复审 AI 的代码。**

AI 写的代码，你一定要看一遍。看不懂的让 AI 解释给你听。不理解的就别用。

### **第四条：学会问对的问题。**

“帮我用 Python 写一个下载文件的脚本”——不如——“帮我用 Python 写一个下载文件的脚本，从 URL 下载到本地指定目录，支持进度显示，支持断点续传。”

### **第五条：保持迭代。**

AI 写第一版，你审核、测试，反馈给 AI 修改。来回几轮，代码质量就上去了。

## **4、AI 编程的未来**

很多人担心：AI 这么强了，程序员是不是要失业了？

阿色的看法：程序员不会失业，但“只会写代码的程序员”可能会被淘汰。

AI 时代，编程的门槛在降低，但“解决问题”的能力更重要。

以前：你得懂语法、懂算法、懂 API 才能写代码。

现在：你只要知道你要啥，AI 帮你写。

未来：你需要的能力不再是“怎么写代码”，而是：

- “要做什么”——分析和拆解需求的能力
- “这么做对不对”——判断和审查 AI 输出的能力
- “出了问题咋办”——排错和 Debug 的能力
- “能不能更好”——优化和迭代的能力

换句话说，程序员的工作从“技术劳动”变成了“脑力劳动+技术劳动”，对综合能力要求更高了。

好消息是：不懂编程的人也能用 AI 做些基础的开发了。这对于非技术人员来说，是巨大的解放。

## 小结一下

- AI 能写代码、解释代码、找 bug、重构、写文档、当老师
- AI 不能保证正确、不能理解业务、不能代替架构决策
- 用好 AI：先想清楚、分解任务、复审代码、问对问题、保持迭代
- 程序员不会失业，但需要的能力在不断升级

## 8.6 OpenClaw 及各种龙虾

### 1、什么是龙虾？

最近爆火的各类“龙虾”，本质都是 AI 智能体框架。

OpenClaw 是开源通用智能体框架，也是所有龙虾产品的底层鼻祖。

它能操控电脑、识别界面、自动执行各类电脑操作，给 AI 装上眼睛和手，不局限于聊天，能实打实本地操作电脑完成任务。

所以它也非常危险。使用它的时候要注意个人信息、隐私、涉密事项、财产等安全问题。

市面上带 Claw 后缀的工具，大家通俗统称为龙虾工具，目前各大互联网公司都在扎堆研发，掀起了“养虾”热潮。

### 2、其他龙虾产品

#### 阿里 QwenPaw：

这本书最开始的初稿就是由 QwenPaw 编写完成。

它前身是阿里开源的 CoPaw，是最早一批基于 OpenClaw 开发的本土化智能体，深度适配通义千问模型，主打本地部署、数据安全，操作简洁、上手门槛低，偏向日常办公、简单自动化操作，也是大众接触最多的入门级龙虾。

### 腾讯 WorkBuddy:

腾讯推出的桌面智能龙虾工具，深度贴合微信、腾讯办公生态，主打办公自动化、文件整理、日常辅助操作，适配国内办公场景，联动腾讯全系软件，办公兼容性拉满。

### 智谱 AutoClaw:

主打一键本地安装，开箱即用，内置大量现成技能，适配零基础用户，偏向内容创作、简单批量操作，通用性强，无需复杂配置就能上手。

### 其他常见龙虾:

除此之外，还有字节、百度等大厂推出的同类智能体工具，比如百度红手指、HiClaw 等。所有龙虾共性都是通用型智能体，功能杂、覆盖面广，主打全能通用，适配多种日常电脑操作场景。

## 3、龙虾适不适合做 AI 编程？

直白说：绝大多数龙虾不适合专业 AI 编程。

龙虾的优势是通用全能，能操作桌面、处理文件、完成简单自动化任务，但短板也很明显：编程专业性不足、代码逻辑优化差、复杂项目适配度低，还容易出现冗余代码、逻辑漏洞，甚至误删文件、错误执行指令。

它们主打综合通用，不是为编程量身打造。

## 4、编程该怎么选工具？

写代码、做开发，不要盲目跟风用各类龙虾。

编程最好选择专门化的编程智能体。

这类专业工具聚焦代码场景，适配开发逻辑，语法精准、bug 排查能力强，支持项目重构、工程优化、代码调试，专门针对编程场景深度优化，比通用龙虾靠谱得多。

总结：龙虾适合玩一玩、做办公自动化、简单杂活；专业编程开发，优先选专精编程 AI，不要混用通用龙虾工具。



## 本章小结

这一章，咱们唠了 AI 和大语言模型的事儿：

AI 是让机器干脑力活儿的工具，本质是“找规律机器”而不是“思考者”。

大语言模型的核心是预测下一个词，靠海量数据和巨大算力训练出来的。

LLM 有很多：ChatGPT、Claude、Gemini、DeepSeek、通义千问、文心一言、豆包.....各有特色。

提示词是跟 AI 交流的关键，说得越清楚，AI 干得越好。

AI 编程很强大，但不是万能的。你得会审核、会判断、会迭代。

龙虾是功能强大的智能体，但它不一定适合做 AI 编程。

学会了这些，你就能更从容地用好 AI，而不是被 AI 带着跑。

下一章，咱们唠唠编程工具的那点事儿。

# 第九章：编程工具的那点事儿

## 开场白

前面几章讲了计算机、程序、网络、部署、运维这些概念，这一章咱们唠唠编程工具。

工欲善其事，必先利其器。工具选好了、环境配好了，写代码才能顺手。

这一章，咱们唠唠你需要哪些工具、怎么选、怎么用。

## 9.1 编程环境是啥玩意儿？

### 1、啥是编程环境？

编程环境就是你写代码、跑代码的地方。

编程环境的组成：

- 操作系统：Windows/Mac/Linux
- 运行环境：Python/Node.js/Java 等
- 编辑器：VS Code/PyCharm 等
- 命令行终端：Terminal
- 版本控制：Git
- 调试和分析工具：IDE 内置调试器/Chrome DevTools/pprof 等

### 2、环境咋配？

不同语言、不同项目，环境配置不一样。但大体的步骤是：

#### 第一步：安装操作系统

如果你是新手，用 Windows 就行，熟悉的界面，操作方便。

如果你想深入学编程，可以试试 Linux 或 macOS。

#### 第二步：安装运行环境

根据你要学的语言，安装对应的运行环境：

去官网下载安装包，一路下一步就行，很简单。

#### 第三步：安装编辑器

推荐 VS Code，免费、插件丰富、跨平台，新手老手都用它。阿色一直都用它。

#### 第四步：安装 Git

版本控制必备，后面会细说。

#### 第五步：安装包管理器

Python 自带 pip，Node.js 自带 npm，一般不用单独装。

### 3、环境变量是啥？

安装完运行环境，有时候还要配置环境变量。

啥是环境变量？

环境变量就是操作系统记录的一些全局变量，程序运行时读取这些变量。

最常见的场景：

你装了 Python，但命令行输入 python 提示找不到命令。

为啥？因为操作系统不知道 Python 装在哪。

你需要把 Python 的安装路径添加到 PATH 环境变量里，操作系统才能找到它。

Windows 怎么配环境变量：

- 1) 右键此电脑 → 属性
- 2) 点击高级系统设置
- 3) 点击环境变量
- 4) 在系统变量里找到 Path，点击编辑
- 5) 点击新建，添加路径
- 6) 一路确定，重启命令行生效

Mac/Linux 怎么配：

编辑 ~/.bashrc 或 ~/.zshrc 文件：

```
export PATH="/path/to/python:$PATH"
```

然后执行：

```
source ~/.bashrc
```

### 小结一下

- 编程环境包括操作系统、运行环境、编辑器、命令行等
- 安装步骤：系统 → 运行环境 → 编辑器 → Git → 包管理器
- 环境变量告诉操作系统程序在哪，有时需要手动配置

## 9.2 代码编辑器：VS Code 为啥这么火？

### 1、啥是代码编辑器？

代码编辑器就是写代码的软件，就像写文档用 Word，写代码用编辑器。

你可能会问：记事本不能写代码吗？

能，但太原始了。没有语法高亮、没有智能提示、没有自动补全.....写起来累死人。

代码编辑器就是为写代码优化的，有各种贴心功能。

### 2、常见编辑器

记事本（Notepad）：系统自带，啥也没有

Notepad++：轻量，只有 Windows 版

VS Code：目前最火，免费开源

Sublime Text：轻量快速，收费

Vim/Emacs：老牌编辑器，学习曲线陡峭

### 3、IDE 和编辑器的区别

你可能还听说过 IDE（Integrated Development Environment，集成开发环境）。

编辑器：主要功能是写代码，轻量、灵活。插件丰富，想要啥功能装啥。

IDE：功能更全面，集成了编辑、调试、编译、运行等功能。重量级，开箱即用。

选哪个？

- 新手：VS Code，轻量好上手
- Python 开发：VS Code 或 PyCharm
- Java 开发：IntelliJ IDEA
- 前端开发：VS Code
- 不想折腾：IDE（开箱即用）

## 4、VS Code 为啥这么火？

免费开源

微软出品，不花钱就能用，而且开源。

插件丰富

各种语言的插件、主题插件、功能插件，想装啥装啥。插件也叫扩展。

常用的插件：

- Python: Python 插件（微软官方）
- JavaScript/TypeScript: ES7+ React/Redux 等
- Git: 内置 Git 支持
- AI 编程: 通义灵码、GitHub Copilot、Claude Code 等
- 中文: 中文语言包

跨平台

Windows、macOS、Linux 都能用，体验一致。

## 5、怎么用 VS Code？

安装：去官网 [code.visualstudio.com](https://code.visualstudio.com) 下载安装包，一路下一步。

基本使用：

- 1) 打开文件夹：File → Open Folder
- 2) 新建文件：Ctrl+N
- 3) 保存文件：Ctrl+S
- 4) 打开终端：Ctrl+`（反引号）
- 5) 安装插件：点击左侧扩展图标，搜索插件，点击安装

常用快捷键：

- Ctrl+Shift+P: 打开命令面板
- Ctrl+P: 快速打开文件
- Ctrl+`: 打开终端
- Ctrl+Shift+`: 新建终端
- Alt+↑/↓: 移动当前行
- Ctrl+/: 注释/取消注释

## 小结一下

- 代码编辑器就是写代码的软件，比记事本强大得多
- IDE 功能全面但重，编辑器轻量灵活
- VS Code 免费、开源、插件丰富、跨平台，是目前最火的编辑器
- 常用快捷键能大幅提升效率

## 9.3 终端和命令行：别怕黑窗口

### 1、啥是终端？

终端（Terminal）就是一个黑乎乎的窗口，你在里面输入命令，计算机执行。

很多新手看到终端就害怕——黑乎乎一片，全是英文，敲不对还报错。

其实终端没那么可怕。你不需要记住所有命令，常用的就那么几个。

为啥要用终端？因为有些事情用鼠标点不了——比如装包、运行脚本、查看日志。终端效率更高。

终端和命令行的关系：

- 终端：软件本身，那个窗口
- 命令行：你输入命令的环境，在终端里运行
- Shell（壳）：命令行的解释器，比如 Bash、Zsh、PowerShell

### 2、打开终端

Windows：

- 1) 按 Win+R，输入 cmd，回车
- 2) 或者搜索命令提示符或 PowerShell
- 3) 或者在 VS Code 里按 Ctrl+`

macOS：

- 1) 打开启动台→其他→终端

2) 或按 `Cmd+空格`，搜索终端

Linux:

按 `Ctrl+Alt+T`

### 3、常用命令

文件操作:

- `ls`: 列出当前目录的文件 (Linux/Mac)
- `dir`: 列出当前目录的文件 (Windows)
- `cd 目录名`: 进入某个目录
- `cd ..`: 回到上级目录
- `mkdir 目录名`: 创建新目录
- `rm 文件名`: 删除文件 (Linux/Mac)
- `del 文件名`: 删除文件 (Windows)
- `cp 源 目标`: 复制文件
- `mv 源 目标`: 移动/重命名文件

查看文件:

- `cat 文件名`: 查看文件内容 (Linux/Mac)
- `type 文件名`: 查看文件内容 (Windows)

其他:

- `pwd`: 显示当前目录路径
- `clear`: 清屏 (Linux/Mac)
- `cls`: 清屏 (Windows)
- `echo 文字`: 输出文字

### 4、路径是啥?

路径就是文件或文件夹在电脑上的地址。

绝对路径: 从根目录开始写。比如:

Windows: `C:\Users\你的名字\Documents`

Mac/Linux: `/Users/你的名字/Documents`

相对路径：从当前位置开始写。比如：

如果你在 `Documents` 目录下，要访问 `test.txt`，直接写 `test.txt` 就行。

`./` 表示当前目录，`../` 表示上级目录。

几个特殊符号：

- `./`：当前目录
- `../`：上级目录
- `~`：用户主目录（Linux/Mac）

## 5、小技巧

按 `Tab` 键自动补全命令和路径

按 `↑↓` 键切换历史命令

按 `Ctrl+C` 终止正在运行的命令

多个命令可以用 `&&` 连接：`cmd1 && cmd2`

命令太长可以用 `\` 换行继续写

## 小结一下

- 终端是黑窗口，在里面输入命令让计算机干活
- 常用命令不多：`cd`、`ls/dir`、`mkdir`、`rm/del`、`cp`、`mv`
- 路径分绝对和相对，`.`是当前目录，`..`是上级目录
- `Tab` 补全、`↑↓`翻历史、`Ctrl+C` 终止，这几个小技巧很实用

## 9.4 Git：版本控制的王者

### 1、啥是版本控制？

版本控制就是记录文件修改历史的系统。

你写论文的时候是不是这样：



论文\_v1.docx、论文\_v2.docx、论文\_最终版.docx、论文\_最终版 2.docx.....

写代码也是一样——改来改去，想回退找不到之前的版本。

版本控制就是解决这个问题的。它能：

- 记录每次修改：谁、啥时候、改了啥
- 回退到任意历史版本：改坏了也不怕
- 多人协作：各改各的，最后合并
- 分支管理：同时开发多个功能

## 2、Git 是啥？

Git 是目前最流行的版本控制工具，没有之一。

Linux 之父 Linus Torvalds 开发的，原本是为了管理 Linux 内核代码。

Git 的特点：

- 分布式：每个人电脑上都有完整的代码仓库
- 快速：大部分操作都在本地，不依赖网络
- 强大：分支管理非常灵活
- 开源免费：不用钱

## 3、Git 的基本操作

安装 Git：

去 [git-scm.com](https://git-scm.com) 下载安装包，一路下一步。

配置身份：

```
git config --global user.name "你的名字"
```

```
git config --global user.email "你的邮箱"
```

基本流程：

```
初始化仓库：git init
```

```
添加文件到暂存区：git add 文件名 或 git add .（添加所有）
```

提交到仓库：`git commit -m "提交说明"`

查看状态：`git status`

查看历史：`git log`

Git 的核心概念：

- 仓库（Repository）：存放代码的地方
- 暂存区（Staging）：提交前的临时存放区
- 提交（Commit）：一次保存，生成一个版本
- 分支（Branch）：独立的开发线

## 4、Git 的工作流程

单人开发：

`git init` → 写代码 → `git add` → `git commit` → 重复

多人协作：

克隆远程仓库：`git clone` 仓库地址

创建分支：`git branch` 分支名

切换到分支：`git checkout` 分支名

在分支上修改、提交

合并到主分支：`git merge` 分支名

推送到远程：`git push`

拉取最新代码：`git pull`

## 5、GitHub 是啥？

GitHub 是一个代码托管平台，帮你把代码存到云端。

它提供：

- 代码托管：把你的代码存在上面
- 协作开发：多人一起开发项目
- 开源社区：很多开源项目在上面
- 其他功能：Issue（问题跟踪）、Wiki、Actions（自动化）等

类似平台：

平台	特点
GitHub	全球最大，被微软收购
GitLab	可以自己搭建
Gitee	国内平台，访问快
Bitbucket	Atlassian 出品，与 Jira 集成好

## 小结一下

- 版本控制帮你记录每次修改，随时可以回退
- Git 是最流行的版本控制工具，分布式、快速、强大
- 基本概念：仓库、提交、分支、暂存区、远程仓库
- 基本操作：init、add、commit、push、pull
- GitHub 是托管 Git 仓库的平台，也是程序员社区

## 9.5 AI 编程助手：你的随身参谋

### 1、啥是 AI 编程助手？

AI 编程助手是用 AI 帮你写代码的工具。

你描述需求，AI 给你代码；你写了一半，AI 帮你补全；你报错了，AI 帮你分析。

就像有个参谋在旁边，你不懂的可以问它。

## 2、国外 AI 编程助手

### GitHub Copilot

微软出品，集成在 VS Code 里。

- 自动补全：写代码时自动提示
- 根据注释生成代码：写个注释，AI 给你代码
- 收费，但有免费试用

### Cursor

AI 原生编辑器，基于 VS Code 改的。

- 深度集成 AI，能在编辑器里直接跟 AI 对话
- 免费版有限制，付费版功能更强

### Claude Code

强大的 AI 智能体。阿色主要使用它。

Anthropic 出品，命令行工具，可以以插件形式嵌入到 VS Code。

- 能帮你写代码、改 bug、解释代码
- 还能做环境部署、系统安装、日常办公等其他工作
- 需要大语言模型 API 密钥支持

## 3、国内 AI 编程助手

### 通义灵码

阿里出品，免费。

- 支持多种语言
- 集成在 VS Code 和 JetBrains IDE 里

### 豆包 MarsCode

字节跳动出品，免费。

- 支持多种语言
- 有 VS Code 插件，也有在线 IDE

### CodeGeeX

清华出品，免费开源。

- 支持多种语言
- 有 VS Code 插件

## TRAE

- 字节跳动出品，个人版永久免费。
- AI 原生 IDE，支持 IDE 模式（AI 辅助）与 SOLO 模式（AI 自主完成）
- 支持多语言、设计稿转代码、自动调试与修复
- 有独立客户端（Windows/macOS/Linux）、在线 IDE，也可作为 VS Code 插件

## 4、怎么用 AI 编程助手？

### 安装插件

以 VS Code 为例：

- 1) 打开 VS Code
- 2) 点左侧"扩展（俗称插件）"图标
- 3) 搜索想要使用的插件，如：'Claude Code'、'通义灵码'、'Copilot'等
- 4) 点"安装"
- 5) 按提示登录或配置

### 使用方式

- 自动补全：写代码时 AI 自动提示，按`Tab`接受
- 聊天：在侧边栏跟 AI 对话，问问题
- 代码解释：选中代码，让 AI 解释
- 生成代码：描述需求，让 AI 生成代码

## 5、使用建议

AI 给的代码，别直接复制粘贴

先看懂，理解每行在干啥，再用。

AI 也会犯错

尤其是一些细节、边界情况，AI 可能处理不好，要检查。

## 学会提问

把背景、需求、约束说清楚，AI 才能给好答案。

## 把 AI 当参谋，不是老板

AI 给你建议，你来判断要不要采纳。

## 小结一下

- AI 编程助手能帮你写代码、改 bug、解释代码
- 国外：GitHub Copilot、Cursor、Claude Code
- 国内：通义灵码、豆包 MarsCode、CodeGeeX
- AI 给的代码要检查，别盲目相信

## 本章小结

这一章，咱们唠了编程工具：

编程环境包括操作系统、运行环境、编辑器、命令行等，配好了才能顺畅写代码。

VS Code 是最火的编辑器，免费、插件丰富、轻量快速。

命令行没那么吓人，学会几个常用命令就行。

Git 是版本控制的王者，帮你记录修改、协作开发。

AI 编程助手能帮你写代码，国内外都有好用的工具，选一个顺手的就行。

下一章，咱们唠唠怎么跟 AI 打交道。

# 第十章：与 bug 作斗争的那点事儿

## 开场白

用 AI 写代码，难免会出问题。

报错了，跑不通，结果不对.....这些问题，每个程序员都会遇到。

这一章，咱们唠唠怎么排查问题，怎么解决问题。

别慌，有问题就有办法。

## 10.1 报错信息咋看？

### 1、报错信息是啥？

报错信息，就是程序告诉你：出问题了。

报错信息一般包含：

- 错误类型：啥类型的错误
- 错误位置：在哪个文件的哪一行
- 错误原因：为啥出错了

### 2、常见的错误类型

语法错误（Syntax Error）

代码写错了，程序根本跑不起来。

比如括号没闭合、冒号漏写了、缩进错了。

### 【错误示例】

```
if x > 0
    print("正数") # if 后面忘了冒号
```

对于中国人而言，要特别注意标点符号半角和全角的问题，特别是分号和冒号。编程使用的标点符号都是半角的，写全角的就会报语法错误。

### 运行时错误 (Runtime Error)

代码语法没问题，但跑的时候出错了。

比如除以零、访问不存在的索引、打开不存在的文件。

### 【错误示例】

```
x = 10 / 0 # 除以零
```

### 逻辑错误 (Logic Error)

代码能跑，但结果不对。

这是最难查的，因为程序不报错，你得自己发现结果不对。

**【错误示例】：**想算平均值，但写错了

```
average = a + b / 2 # 应该是 (a + b) / 2
```

## 3、怎么看报错信息？

### 第一步：看错误类型

先搞清楚是啥类型的错误，心里就有谱了。

### 第二步：看错误位置

找到出错的文件和行号，去那行看看。

### 第三步：看错误原因

理解为啥出错，才能改对。

### 第四步：看上下文



有时候问题不在报错的那一行，在前面几行。

## 4、实例：看一个 Python 报错

```
Traceback (most recent call last):  
  File "test.py", line 5, in <module>  
    result = divide(10, 0)  
  File "test.py", line 2, in divide  
    return a / b
```

ZeroDivisionError: division by zero

咱们来拆解：

- `File "test.py", line 5`：在 test.py 的第 5 行调用 divide 函数
- `File "test.py", line 2`：在 test.py 的第 2 行，a / b 出错了
- `ZeroDivisionError: division by zero`：除以零错误

这下就知道问题在哪了：除数是零，不能除。

## 小结一下

- 报错信息告诉你问题出在哪
- 先看类型，再看位置，再看原因
- 语法错误最容易改，逻辑错误最难找

## 10.2 常见错误类型

### 1、TypeError：类型错误

操作的对象类型不对。

#### 【错误示例】

```
"hello" + 123  # 字符串不能直接加数字
```

解决方法：转换类型。

```
"hello" + str(123) # 把数字转成字符串
```

## 2、NameError: 名字错误

用了一个不存在的变量或函数。

### 【错误示例】

```
print(username) # username 没定义过
```

解决方法：检查变量名，是不是拼写错了，是不是没定义。

## 3、IndexError: 索引错误

访问了不存在的索引。

### 【错误示例】

```
list = [1, 2, 3]
print(list[5]) # 列表只有 3 个元素，没有索引 5
```

解决方法：检查列表长度，确保索引在范围内。

## 4、KeyError: 键错误

访问了不存在的字典键。

### 【错误示例】

```
person = {"name": "张三"}
print(person["age"]) # 没有 age 这个键
```

解决方法：用 `get` 方法，或者先检查键是否存在。

```
print(person.get("age", "未知")) # 如果没有，返回"未知"
```

## 5、FileNotFoundError: 文件不存在错误

要打开的文件找不到。

### 【错误示例】

```
with open("data.txt") as f: # data.txt 不存在
    content = f.read()
```

解决方法：检查文件路径，确保文件存在。

## 6、AttributeError：属性错误

对象没有这个属性或方法。

### 【错误示例】

```
num = 123
num.append(4) # 数字没有 append 方法
```

解决方法：检查对象类型，确认它有这个方法。

## 7、IndentationError：缩进错误

Python 对缩进很敏感，缩进不对就报错。

### 【错误示例】

```
if x > 0:
print("正数") # 没有缩进
```

解决方法：统一用 4 个空格缩进，别混用空格和 Tab。

## 小结一下

- TypeError：类型不对
- NameError：变量没定义
- IndexError：索引越界
- KeyError：字典键不存在
- FileNotFoundError：文件找不到
- AttributeError：属性或方法不存在
- IndentationError：缩进不对

## 10.3 Debug 的基本招数

### 1、啥是 Debug?

Debug，就是找 bug、改 bug。

Bug，就是程序里的错误。

Debug 的方法有很多，咱们说几个常用的。

### 2、打印调试

最简单的方法，就是打印。

在可能出问题的地方，打印变量的值，看看是不是你预期的。

```
x = 10
y = 0
print(f"x = {x}, y = {y}") # 打印看看
result = x / y # 这里会报错
```

打印调试的优点：

- 简单，不需要额外工具
- 直观，能看到变量的值

缺点：

- 要手动加 `print`，调试完还得删
- 打印多了，输出会很乱

### 3、用调试器

调试器能让你一步步执行代码，查看变量的值。

以 VS Code 为例：

- 1) 在代码行号左边点一下，打红点断点
- 2) 按 F5 开始调试
- 3) 程序会在断点处停下来
- 4) 可以查看变量、一步步执行

调试器的优点：

- 不用改代码
- 能精确控制执行过程
- 能看到所有变量的值

缺点：

- 要学习怎么用
- 对于复杂问题，可能比较耗时

## 4、二分法定位

问题不知道在哪？用二分法。

在代码中间加个打印，看看程序有没有执行到这。

如果执行到了，问题在后半段；没执行到，问题在前半段。

继续二分，直到找到问题所在。

## 5、注释法

把一部分代码注释掉，看问题还在不在。

问题消失了，说明在被注释的代码里；问题还在，说明在其他地方。

## 6、搜索错误信息

把报错信息复制到搜索引擎里搜一下。

很多错误别人都遇到过，网上有解决方案。

## 小结一下

- 打印调试：最简单，加 `print` 看变量
- 用调试器：设断点，一步步执行
- 二分法：从中间开始，缩小范围
- 注释法：注释掉部分代码，定位问题
- 搜索：网上找答案

## 10.4 怎么问 AI?

### 1、问 AI 的姿势

用 AI 编程，最重要的技能之一：会问 AI。

问得好，AI 给你好答案；问得不好，AI 给你瞎答案。

### 2、错误的问法

"我的代码报错了，怎么办？"

这种问题，AI 没法回答，因为它不知道：

- 你的代码是啥
- 报错信息是啥
- 你想干啥

### 3、正确的问法

提供完整的上下文：

- 1) 你想干啥（需求）
- 2) 你的代码是啥
- 3) 报错信息是啥
- 4) 你已经尝试了啥
- 5) 还可以给它发截图

示例：

我想读取一个 CSV 文件，计算某列的平均值。

我的代码：

```
import pandas as pd
df = pd.read_csv("data.csv")
avg = df["price"].mean()
print(avg)
```

报错信息：

**KeyError: "price"**

我已经检查了文件，确实有 price 这一列。

请问问题出在哪？

这样问，AI 就知道怎么帮你了。

## 4、问 AI 的技巧

### 分步骤问

问题很复杂？拆成小问题，一个个问。

### 让 AI 解释

不懂某段代码？让 AI 给你解释。

"请解释这段代码每一行在干啥："

### 让 AI 改错

给 AI 代码和报错信息，让它帮你改。

### 让 AI 给方案

描述需求，让 AI 给你一个解决方案。

## 5、不要完全依赖 AI

AI 也会犯错。

AI 给的代码，你要看懂，确认没问题再用。

特别是安全相关的代码，更要仔细检查。

## 小结一下

- 问 AI 要提供完整上下文：需求、代码、报错、已尝试
- 复杂问题拆成小问题
- 让 AI 解释、改错、给方案
- AI 给的代码要检查，别盲目相信

## 10.5 求助的正确姿势

### 1、啥时候该求助？

自己折腾了一阵子，还是搞不定，就该求助了。

但求助之前，先做几件事：

- 1) 搜索一下，网上有没有答案
- 2) 看看文档，有没有说明
- 3) 问一下 AI，让它帮你分析

### 2、去哪求助？

**Stack Overflow**

程序员问答社区，问题多，答案也多。

注意：提问前先搜索，别问重复的问题。

**GitHub Issues**

开源项目的问题跟踪，如果你用的是开源库，可以去看看 Issues。

**中文社区**

掘金、思否、CSDN 等，也有不少问答。

**技术群**

加几个技术群，问群里的大佬。

### 3、怎么提问？

提问之前，准备好这些：

- 1) 问题描述：你想干啥，结果咋样
- 2) 代码：最小可复现的代码
- 3) 环境：操作系统、语言版本、相关库版本
- 4) 已尝试：你试过啥方法
- 5) 报错信息：完整的报错信息

好的提问示例：



标题：Python 读取 CSV 时报 `KeyError`

环境：

Python 3.10

pandas 2.0

Windows 11

问题描述：

读取 CSV 文件后，访问某列报 `KeyError`。

代码：

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df.columns) # 输出: Index(['name', ' price'],
type='object')
print(df["price"]) # 报错
```

报错信息：

`KeyError: "price"`

已尝试：

确认文件中有 `price` 列

试过 `df.get("price")`，返回 `None`

请帮忙看看问题出在哪？

原来问题是列名前面有个空格，是 `" price"` 而不是 `"price"`。

## 4、提问的礼仪

- 1) 先搜索，再提问
- 2) 标题要清楚，别写“求助”“救命”
- 3) 代码要完整，能复现问题
- 4) 得到答案后，感谢回答者
- 5) 解决后，说一下怎么解决的，帮助后来人

## 小结一下

- 求助前先搜索、看文档、问 AI
- Stack Overflow、GitHub、技术群是好地方
- 提问要说清楚：环境、代码、报错、已尝试

- 有礼貌，得到帮助要感谢

## 本章小结

这一章，咱们唠了出了问题咋整：

报错信息是你的朋友，它能告诉你问题在哪。学会看报错，问题就解决了一半。

常见错误类型要熟悉，看到了就知道大概怎么改。

Debug 方法有很多：打印、调试器、二分法、注释法、搜索。

问 AI 要有技巧，提供完整上下文，AI 才能帮你。

求助要讲究方式，问题说清楚，别人才能帮你。

下一章，咱们唠唠安全和法律那些事儿。

# 第十一章：安全和法律的那点事儿

## 开场白

用 AI 写代码，除了技术问题，还有安全和法律问题。

这一章，咱们唠唠这些"非技术"但很重要的事儿。

别觉得这些不重要，出了事可就麻烦了。

## 11.1 代码安全基本意识

### 1、啥是代码安全？

代码安全，就是你的代码不会被坏人利用，不会泄露信息，不会造成损失。

安全问题无处不在，稍不注意就会踩坑。

### 2、常见的安全问题

#### 敏感信息泄露

把密码、密钥、token 等写死在代码里，代码泄露了，这些信息也泄露了。

【错误示例】：密码写死在代码里

```
password = "my_secret_password_123"
```

#### SQL 注入

用户输入直接拼到 SQL 语句里，黑客可以构造输入来执行恶意 SQL。

**【错误示例】：**SQL 注入

```
sql = "SELECT * FROM users WHERE name = '{user_input}'"
```

用户输入：' OR '1'='1

实际执行的 SQL：SELECT \* FROM users WHERE name = '' OR '1'='1'

会返回所有用户！

**跨站脚本（XSS）**

用户输入直接显示在网页上，黑客可以注入恶意脚本。

**【错误示例】：**XSS

```
html = "<div>{user_input}</div>"
```

用户输入：<script>alert('我被黑了')</script>

网页会执行这段脚本！

**文件上传漏洞**

允许用户上传文件，但没有检查文件类型和内容，黑客可以上传恶意脚本。

### 3、安全编码的基本原则

**最小权限原则**

程序只获取它需要的最小权限，不多要。

**永远不相信用户输入**

用户输入的数据，都要检查、过滤、转义。

**敏感信息要保护**

密码要加密存储，密钥要放安全的地方，不要写死在代码里。

**使用安全的库和函数**

不要自己写加密算法，用成熟的库。

## 及时更新依赖

依赖库有漏洞时，要及时更新。

## 4、AI 写的代码安全吗？

AI 写的代码，可能有安全问题。

AI 不知道你的业务场景，不知道安全要求，它只是按常见写法给你生成代码。

所以，AI 给的代码，尤其是涉及安全的代码，要仔细审查。

## 小结一下

- 安全问题无处不在
- 常见问题：信息泄露、SQL 注入、XSS、文件上传漏洞
- 安全原则：最小权限、不信任输入、保护敏感信息
- AI 写的代码要审查安全

## 11.2 保护你的代码和数据

### 1、代码要保护啥？

知识产权：你写的代码，是你的成果，不想被别人拿去用。

商业机密：代码里有你公司的业务逻辑，泄露了不好。

敏感信息：代码里有密码、密钥等，泄露了更麻烦。

### 2、怎么保护代码？

#### 版本控制

用 Git 管理代码，可以追溯修改记录，也可以回滚。

#### 私有仓库

敏感代码放私有仓库，不公开。

## 代码混淆

对重要代码进行混淆，让人看不懂。

## 代码审查

团队成员互相审查代码，发现潜在问题。

## 3、数据要保护啥？

用户数据：用户的信息，要保护好，泄露了是大事。

业务数据：公司的业务数据，是重要资产。

日志数据：日志里可能有敏感信息，也要保护。

## 4、怎么保护数据？

### 加密存储

敏感数据加密后存储，即使数据库泄露，也看不到明文。

### 访问控制

谁能访问什么数据，要控制好。

### 数据备份

重要数据要备份，丢了能恢复。

### 数据脱敏

非生产环境用脱敏数据，不暴露真实信息。

## 5、云服务安全

现在很多代码和数据在云上，云安全也很重要。

### 选择可靠的云服务商

大厂的安全措施更完善。

### 配置好权限

云服务的权限要配置好，别开太大。

## 开启日志和监控

有问题能及时发现。

## 定期检查

定期检查云资源的配置，有没有安全隐患。

## 小结一下

- 代码要保护：知识产权、商业机密、敏感信息
- 数据要保护：用户数据、业务数据、日志数据
- 保护方法：加密、访问控制、备份、脱敏
- 云服务也要注意安全

## 11.3 开源协议和版权

### 1、啥是开源协议？

开源协议是法律文件，规定别人可以怎么用你的代码。

不是所有开源代码都可以随使用，不同的协议有不同的要求。

### 2、常见的开源协议

#### MIT 协议

最宽松的协议。

- 可以随使用、随便改、随便分发
- 只要保留版权声明就行

#### Apache 协议

比 MIT 严格一点。

- 可以用、可以改、可以分发
- 要保留版权声明和许可证

- 修改后要说明改了啥

## GPL 协议

传染性协议。

- 可以用、可以改、可以分发
- 如果你用了 GPL 的代码，你的代码也要开源
- 这是 GPL 的特点：开源要"传染"

## BSD 协议

类似 MIT，很宽松。

- 可以用、可以改、可以分发
- 保留版权声明

## LGPL 协议

比 GPL 宽松。

- 可以动态链接 LGPL 库，你的代码不用开源
- 如果你修改了 LGPL 库本身，修改部分要开源

## 3、怎么选开源协议？

- 想让别人随使用：MIT、BSD
- 想让别人用但保留专利权：Apache
- 想让你的代码永远开源：GPL

## 4、用开源代码要注意啥？

### 看清楚协议

用别人的开源代码前，看清楚是什么协议，你能不能用。

### 遵守协议要求

协议要求保留版权声明就保留，要求开源就开源。

### 不要瞎抄

即使开源代码，也不要直接复制粘贴，要理解后用自己的方式写。



## 小结一下

- 开源协议规定怎么用开源代码
- MIT、Apache、GPL 是常见协议
- MIT 最宽松，GPL 有传染性
- 用开源代码要看清协议、遵守要求
- AI 生成代码的版权还不明确

## 11.4 AI 编程的法律风险

### 1、AI 生成内容的版权问题

AI 生成的内容，版权归谁？

这个问题，法律界还在讨论。

目前的主流观点：

- AI 生成的内容，版权归属不明确
- 人对 AI 生成内容进行修改后，人对修改部分可能有版权
- 用 AI 生成的内容，有侵权风险

### 2、AI 可能"抄"别人的代码

AI 是用大量代码训练的，可能学到了别人的代码。

如果 AI 给你的代码和别人的一样，你可能就侵权了。

### 3、怎么降低风险？

不要直接用 AI 生成的代码

理解后，用自己的方式写。

检查 AI 生成的代码

搜索一下，看看是不是和别人的代码一样。

重要的代码，自己写

核心代码、商业代码，最好自己写，不要用 AI 生成。

#### 4、AI 使用的合规问题

有些公司、行业，对 AI 使用有限制。

比如：

- 涉密单位不能用 AI 处理敏感信息
- 有些公司禁止把代码发给外部 AI 服务

用 AI 前，要了解你所在环境的规定。

#### 5、AI 服务的隐私问题

你把代码发给 AI 服务，AI 服务能看到你的代码。

如果代码敏感，要考虑这个风险。

可以选择本地部署的 AI，或者公司内部的 AI 服务。

#### 小结一下

- AI 生成内容的版权不明确
- AI 可能"抄"别人的代码，有侵权风险
- 降低风险：理解后重写、检查、核心代码自己写
- 注意 AI 使用的合规和隐私问题

## 11.5 职业道德和责任

### 1、程序员的职业道德

写代码不只是技术活，也有道德要求。

不作恶

不要写恶意代码，不要侵犯别人隐私，不要做违法的事。

对用户负责

你写的代码影响用户，要对用户负责。

### **诚实**

不懂就学，不要装懂；有问题就说，不要隐瞒。

### **尊重他人**

尊重同事、用户、开源贡献者。

## **2、AI 编程的道德问题**

### **不要用 AI 作弊**

比如用 AI 写作业、写考试答案，这不道德。

### **不要用 AI 作恶**

比如用 AI 写攻击代码、钓鱼网站，这是违法的。

### **对 AI 生成的内容负责**

你用 AI 生成了代码，你就要对这段代码负责，不能出了问题怪 AI。

## **3、出了问题谁负责？**

你用 AI 写的代码，出了问题，谁负责？

当然是你负责。

AI 只是工具，你是使用者，你要对你的代码负责。

### **小结一下**

- 程序员要有职业道德：不作恶、负责、诚实、尊重
- AI 编程也要有道德：不作弊、不作恶、对 AI 生成内容负责
- 出了问题，使用者负责

## 本章小结

这一章，咱们唠了安全和法律那些事儿：

代码安全要有意识，敏感信息别泄露，用户输入别信任，AI 代码要审查。

保护代码和数据，用加密、访问控制、备份等方法。

开源协议要看清，不同协议要求不同，遵守协议要求。

AI 编程的法律风险，版权不明确、可能侵权、有合规和隐私问题。

职业道德和责任，不作恶、负责、诚实、尊重，对 AI 生成的代码负责。

到这里，这本书的内容就差不多了。

# 附录

## 附录 A：常用术语表（中英文对照）

### 一、计算机基础

中文术语	英文术语	解释
中央处理器	CPU (Central Processing Unit)	计算机的大脑，负责执行指令和运算
内存	Memory / RAM (Random Access Memory)	程序运行时临时存放数据的地方，断电后数据丢失
硬盘	Hard Disk / HDD / SSD	永久存储数据的设备，断电后数据不丢失
固态硬盘	SSD (Solid State Drive)	用闪存技术存储的硬盘，速度更快
图形处理器	GPU (Graphics Processing Unit)	专门处理图形和并行计算的处理器
主板	Motherboard	连接所有硬件的核心电路板
显卡	Graphics Card	处理图形输出的硬件
操作系统	Operating System (OS)	管理计算机硬件和软件资源的系统软件
内核	Kernel	操作系统的核心部分
驱动程序	Driver	让操作系统识别和使用硬件的软件

### 二、编程语言

中文术语	英文术语	解释
编程语言	Programming Language	人用来写程序的语言
源代码	Source Code	程序员写的代码
机器码	Machine Code	计算机能直接执行的代码
编译	Compilation	把源代码翻译成机器码的过程

解释	Interpretation	边执行边翻译代码的方式
编译器	Compiler	执行编译的程序
解释器	Interpreter	执行解释的程序
语法	Syntax	编程语言的语法规则
语义	Semantics	代码的含义和逻辑
标识符	Identifier	变量、函数等的名称
关键字	Keyword	编程语言保留的专用词
注释	Comment	不被执行的说明文字

### 三、数据类型与结构

中文术语	英文术语	解释
变量	Variable	存储数据的容器
常量	Constant	值不能改变的量
数据类型	Data Type	数据的种类，如整数、字符串等
整数	Integer	没有小数部分的数字
浮点数	Float / Floating Point Number	有小数部分的数字
字符串	String	文本数据
布尔值	Boolean	真或假（True/False）
数组	Array	存储多个相同类型数据的结构
列表	List	有序的可变数据集合
元组	Tuple	有序的不可变数据集合
字典	Dictionary	键值对的数据结构
集合	Set	不重复元素的无序集合
对象	Object	包含属性和方法的数据结构
类	Class	创建对象的模板
实例	Instance	类的具体实例化对象
空	Null / None	表示没有值

### 四、程序结构

中文术语	英文术语	解释
函数	Function	打包一段可重复使用的代码
方法	Method	对象中的函数
参数	Parameter	函数定义时接收的变量
实参	Argument	调用函数时传入的值
返回值	Return Value	函数执行后返回的结果
条件语句	Conditional Statement	根据条件执行不同代码
循环	Loop	重复执行一段代码

迭代	Iteration	循环的一次执行
递归	Recursion	函数调用自身
异常	Exception	程序运行时的错误
异常处理	Exception Handling	捕获和处理异常的机制

## 五、软件工程

中文术语	英文术语	解释
模块	Module	一个独立的代码文件
包	Package	一组相关的模块
库	Library	提供特定功能的代码集合
框架	Framework	提供基础结构的代码框架
接口	Interface	模块之间交互的规范
应用程序接口	API (Application Programming Interface)	程序之间交互的桥梁
软件开发工具包	SDK (Software Development Kit)	开发用的工具集合
集成开发环境	IDE (Integrated Development Environment)	集成编辑、编译、调试的开发环境
版本控制	Version Control	管理代码版本的系统
仓库	Repository	存放代码的地方
分支	Branch	代码的独立开发线
合并	Merge	把不同分支的代码合并在一起
提交	Commit	保存代码修改记录
推送	Push	把本地代码推送到远程仓库
拉取	Pull	从远程仓库获取代码

## 六、网络与 Web

中文术语	英文术语	解释
网络	Network	多台计算机互联的系统
互联网	Internet	全球范围的计算机网络
协议	Protocol	网络通信的规则
IP 地址	IP Address	网络设备的唯一标识
域名	Domain Name	网站的好记名字
域名系统	DNS (Domain Name System)	把域名翻译成 IP 地址的系统
端口	Port	网络服务的入口
防火墙	Firewall	网络安全屏障
代理服务器	Proxy Server	转发请求的中间服务器

超文本传输协议	HTTP (HyperText Transfer Protocol)	网页传输协议
安全超文本传输协议	HTTPS	加密的 HTTP 协议
前端	Frontend	用户看到的界面部分
后端	Backend	服务器端的程序
全栈	Full Stack	前端和后端都懂
网页	Web Page	浏览器显示的页面
网站	Website	多个网页的集合
服务器	Server	提供服务的计算机或程序
客户端	Client	请求服务的程序或设备

## 七、数据库

中文术语	英文术语	解释
数据库	Database	有组织存储数据的系统
关系型数据库	Relational Database	用表格存储数据的数据库
非关系型数据库	NoSQL Database	不用表格存储的数据库
结构化查询语言	SQL (Structured Query Language)	操作关系型数据库的语言
表	Table	数据库中存储数据的表格
字段	Field	表中的列
记录	Record	表中的一行数据
主键	Primary Key	唯一标识记录的字段
外键	Foreign Key	关联其他表的字段
索引	Index	加速查询的数据结构
事务	Transaction	一组操作的执行单元
对象关系映射	ORM (Object-Relational Mapping)	把对象映射到数据库的技术

## 八、云计算与部署

中文术语	英文术语	解释
云计算	Cloud Computing	通过网络提供计算服务
云服务	Cloud Service	云服务商提供的服务
基础设施即服务	IaaS (Infrastructure as a Service)	提供虚拟硬件的云服务
平台即服务	PaaS (Platform as a Service)	提供开发平台的云服务
软件即服务	SaaS (Software as a Service)	提供软件的云服务
部署	Deployment	把程序发布到服务器
容器	Container	隔离运行程序的技术
容器编排	Container Orchestration	管理多个容器的技术



虚拟机	Virtual Machine (VM)	模拟的计算机系统
负载均衡	Load Balancing	分配请求到多台服务器
内容分发网络	CDN (Content Delivery Network)	分布式内容加速网络
虚拟专用网络	VPN (Virtual Private Network)	安全的加密网络连接

## 九、调试与错误

中文术语	英文术语	解释
错误	Error	程序中的问题
异常	Exception	程序运行时的错误
漏洞	Bug	程序中的缺陷
调试	Debug	找 bug、改 bug 的过程
断点	Breakpoint	程序暂停执行的位置
单步执行	Step Execution	一步步执行程序
堆栈跟踪	Stack Trace	函数调用链的记录
日志	Log	程序运行的记录
语法错误	Syntax Error	代码语法不对
运行时错误	Runtime Error	运行时发生的错误
逻辑错误	Logic Error	代码逻辑不对

## 十、安全与法律

中文术语	英文术语	解释
信息安全	Information Security	保护信息不被非法访问
加密	Encryption	把数据变成密文
解密	Decryption	把密文还原成明文
认证	Authentication	验证身份的过程
授权	Authorization	授予访问权限
会话	Session	用户与服务器的交互过程
令牌	Token	认证用的凭证
跨站脚本攻击	XSS (Cross-Site Scripting)	注入恶意脚本的攻击
SQL 注入	SQL Injection	注入恶意 SQL 的攻击
开源	Open Source	源代码公开，可自由使用
许可证	License	使用软件的法律条款
版权	Copyright	作品的法律所有权
知识产权	Intellectual Property	智力成果的法律权利

十一、AI 与机器学习

中文术语	英文术语	解释
人工智能	AI (Artificial Intelligence)	模拟人类智能的技术
机器学习	Machine Learning	让机器从数据中学习
深度学习	Deep Learning	用神经网络的学习方法
神经网络	Neural Network	模拟人脑的计算模型
大语言模型	LLM (Large Language Model)	大规模的语言处理模型
自然语言处理	NLP (Natural Language Processing)	处理人类语言的技术
提示词	Prompt	给 AI 的指令或问题
微调	Fine-tuning	调整模型适应特定任务
训练	Training	用数据教模型学习
推理	Inference	用训练好的模型预测

十二、其他常用术语

中文术语	英文术语	解释
算法	Algorithm	解决问题的步骤
复杂度	Complexity	算法的时间或空间代价
性能	Performance	程序的运行效率
优化	Optimization	提升程序性能
重构	Refactoring	改进代码结构
测试	Testing	检验程序是否正确
单元测试	Unit Test	测试单个函数或模块
集成测试	Integration Test	测试模块间的协作
文档	Documentation	程序的说明资料
开源社区	Open Source Community	共享代码的开发者群体
缓存	Cache	临时存储加速访问
同步	Synchronous	等待完成再继续
异步	Asynchronous	不等待，继续执行
回调	Callback	完成后调用的函数
事件	Event	发生的某种动作
监听	Listen / Watch	等待事件发生
触发	Trigger	引发事件
渲染	Render	生成显示内容
解析	Parse	分析数据结构
序列化	Serialization	把对象转成可存储格式
反序列化	Deserialization	把存储格式转回对象

## 附录 B：推荐学习资源

### 在线教程

- 菜鸟教程：各种编程语言入门教程  
网址：<https://www.runoob.com/>
- 廖雪峰的官方网站：Python、Git 等教程  
网址：<https://www.liaoxuefeng.com/>
- MDN Web Docs：前端开发文档（HTML/CSS/JS 最权威参考）  
网址：<https://developer.mozilla.org/>
- 官方文档：每种语言、每个库都有官方文档  
例如：Python 是 <https://www.python.org/doc/>，React 是 <https://react.dev/>

### 视频课程

- B 站：有很多编程教学视频  
网址：<https://www.bilibili.com>
- 慕课网：系统性的编程课程  
网址：<https://www.imoooc.com>
- Coursera：国外名校课程  
网址：<https://www.coursera.org>

### 练习平台

- LeetCode：算法练习  
网址：<https://leetcode.cn>（中文社区）
- 牛客网：编程练习和面试题  
网址：<https://www.nowcoder.com>
- Codewars：编程挑战  
网址：<https://www.codewars.com>

### 社区

- Stack Overflow：程序员问答社区  
网址：<https://stackoverflow.com>
- GitHub：开源项目托管平台  
网址：<https://github.com>
- 掘金：中文技术社区

网址: <https://juejin.cn>

- V2EX: 技术讨论社区

网址: <https://www.v2ex.com>

## 附录 C: Windows CMD/PowerShell 常用命令

命令	作用
dir	列出当前目录的文件
cd 目录名	进入目录
cd ..	返回上一级目录
cls	清屏
copy 源 目标	复制文件
move 源 目标	移动文件
del 文件名	删除文件
mkdir 目录名	创建目录
rmdir 目录名	删除目录
type 文件名	显示文件内容
ping 地址	测试网络连接
ipconfig	显示网络配置

## 附录 D: VS Code 常用快捷键

快捷键	作用
Ctrl + S	保存
Ctrl + Z	撤销
Ctrl + Y	重做
Ctrl + C	复制
Ctrl + V	粘贴
Ctrl + X	剪切
Ctrl + F	查找
Ctrl + H	替换
Ctrl + /	注释/取消注释
Ctrl + `	打开/关闭终端

F5	开始调试
F9	设置/取消断点
F10	单步跳过
F11	单步进入

# 后 记

这本书写在这儿，就结束了。

回顾一下，咱们唠了计算机、程序、数据、网络、网页、云、工具、AI 编程、问题排查、安全法律这些事儿。

都是基础知识，不深，但应该够用。

AI 生成这本书的初稿用了 10 分钟，我利用闲暇时间审查、修改，持续了大约三、四天。AI 确实太能干了，但还是要有人做最后的审定、把关和负责。

对了，阿色不保证这本书的内容都是正确的，因为我保证不了，有些东西我这个老程序员也不懂。我也是边写边跟 AI 学。

另外请注意，这本书不太严谨，阐述不求精准，甚至可能存在比较严重的错误。请读者自行把握。

最后再让 AI 帮我把这本书从头到尾看一遍，把把关。

阿色最后这一道把关也要交给 AI 了吗？

不然呢？阿色已经看不动了。我就只能多问几个 AI 吧。

## 写这本书的初衷

我用 AI 编程有一阵子了，发现很多人用 AI 写代码，但不了解基础知识。

遇到报错就懵圈，不知道问题出在哪。

AI 给的代码，不知道在干啥。

出了问题，不知道怎么排查。

所以我想写本书，把这些基础知识讲一讲。

希望能帮到想用 AI 编程的朋友。

其实最重要的，我自己也利用这个机会把这些知识重新复习、学习了一遍，特别是一些较新的知识，我也需要学习掌握。

## 关于 AI 编程的一点思考

AI 确实强大，能帮你写代码、改 bug、解释代码。

但 AI 不是万能的。

不懂基础知识，用 AI 写代码也会很吃力。

AI 给的代码可能有问题，你得能看出来。

AI 报的错误你得能看懂，才能改对。

所以，基础知识还是得了解。

AI 是工具，你是主人。工具再好，主人不会用也不行。

最重要的，即便程序是 AI 生成的，但最终的责任人仍然是你。

你不懂基本的原理、架构，你怎么负责？

## 给读者的话

如果你是新手，希望这本书能帮你入门。

如果你有经验，希望这本书能帮你梳理知识。

如果觉得有用，推荐给朋友。

如果觉得有问题，欢迎批评指正。

## 感谢

感谢 AI 帮我写这本书，当然，我修改了很多，补充了很多。

感谢所有开源项目，让我学到了很多。

感谢读者，能读完这本书。

## 最后

编程是一门手艺，也是一场修行，需要思考，更需要实践。

来吧，先上路走两步！

阿色

2026 年于大庆

全书完