从“学”开始学编程

编程学习的认知与实践入门

关于编程学习的认知手册-引言

课程目标：

提醒： 这不是一本编程指南，而是一本关于学习如何认知的课程

这个课程的目的不是教具体的编程知识，而是企图给你一些建议，用于引导你更加顺利地进入学习轨道，少走弯路，避开误区，快速获得学习的喜悦，从而更加可持续地学习，不断得到获得感，最终建立学成的自信。

只有建立了编程的自信心，才能说【我学会了编程】。

我担心的不是你不够聪明，也不担心你不够努力，而是担心你力气用错了地方，或者使用了最费力的方法，那样你就会事倍功半，遭受挫折，从而陷入焦虑，失去继续学习的勇气。

然而，市面上遍布的书籍里，好像很少有针对这方面的书籍，所以决定自己整理，供大家参考。

在后续的课程中，将主要涵盖以下几类内容：

◆ 建立“学习”与“编程学习”的认知框架，提供学习过程的方法论。

包括树立正确的学习心态，获得自主学习的能力，而不是依赖别人教。

◆ 什么是有助于编程的底层认知方法

编程不仅仅是知道计算机语言的知识就够了，更加需要对世界的正确认知。认知错了，编出来的代码自然就错了。

我也发现有些人对于计算机语言本身已经很了解，可是仍然把代码给写错了，说明了现实世界的合理性和底层的认知的重要性要大于计

算机语言本身。比如类似于鸽笼原理这样的原理，对如何认识数据库的构造很有帮助。

还有一些认知框架，可以指导我们怎样构建完美的代码体系。

比如 Asis， To Be， Gap 的认知框架可以用来做需求定义，也可以用来制定开发计划，亦可以指导你如何规划学习方向，具有非常普遍的意义。我尽量把这样的框架整理出来，供大家参考。

这样的框架其实非常多，你们也可以去挖掘自己的认知框架，形成你们自己的思维风格。

这是一系列课程的第一篇，先从什么叫做学习开始，然后进入什么叫编程学习，以及建立程序的世界的初步而完整的概观。

- 学习与编程学习的区别以及联系
- 理解编程学习的三个基础层面：环境、工具、代码
- 掌握程序在单机与网络环境下的基本执行方式

三、课程结构：

第一部分：建立“学习”与“编程学习”的认知框架

1.1 “学”与“习”的本质区别

孔夫子说过，学而时习之，不亦说乎？是对学习的深入骨髓的描述。

既要学又要习，最终达到喜悦的程度。

学：是知识的输入，是观察、模仿、理解的过程。传统意义上“学”更多是接受已有知识。

习：是技能的输出，是实践、反馈、修正、熟练的过程。强调通过反复行动使知识真正转化为能力。

“学”是建立认知的桥梁，“习”是打通通往能力的路径”，二者缺一不可，单靠“学”会导致纸上谈兵，单靠“习”会变成无头苍蝇。

插图：左侧为“学”——读书的人；右侧为“习”——在写的人，中间箭头写“实践”

1.2 学习是一个闭环，而不是线性过程

一次完整的学习经历包括以下环节：

输入 → 理解 → 尝试 → 错误 → 反馈 → 改进 → 熟练 →（新一轮输入）

这个循环帮助我们从模仿者变成掌握者，从知识消费者变成知识创造者

想一想你是如何学会骑自行车的？大概没人会靠读《骑行指南》掌握了骑车技能。再比如学钢琴，真正让你掌握演奏能力的，并不是乐理课本，而是长时间反复的练习和身体记忆的积累。

编程也是一门偏重实践的课，用在实践的时间要远远大于知识的输入。对这个学习闭环周而复始地实践，才是编程学习的本质。

学习的效果不在于知道了多少知识，而是形成一种对所需知识的探索，思考，应用的综合能力，要把技能融入你的整个身体，五官，到手脚，到大脑。这样你就获取了一种关于编程的元能力，这种能力能够让你在面对任何编程任务都能从容应对，最终建立一种自信！

※元能力 是那些能够让你更有效地获取、运用、发展和驾驭各种具体能力的基础性、通用性、高阶能力。它们是个人在快速变化的世界中保持竞争力、实现持续成长和成功的关键基石。投资发展元能力，就是投资于未来的学习力、适应力和核心竞争力。

正确的编程学习闭环的七个阶段应为：

输入：获取信息，听讲、读书、观看

理解：内化信息，转为自己能理解和操作的模型

以后的课程里会包含怎么读代码，怎么读仕样的内容。

尝试：写代码、实践演练。其实，代码不是写出来的，而是调试出来的，调试的时间，远远多于用于编码的时间。

错误：自然出现的问题或失败。谁都不可能写代码一蹴而就。

分享一个我的故事。我曾把代码写下，未经调试直接提交，然后狂妄地告诉测试人员，说，只要发现一个 Bug，罚款 100 日元，结果，我的凄惨可想而知。。。。。

反馈：得到错误原因或外部评价

后续有关于如何调查，如何有效提出问题，如何面对指摘的课程，用来帮助你得到正确的正向反馈。

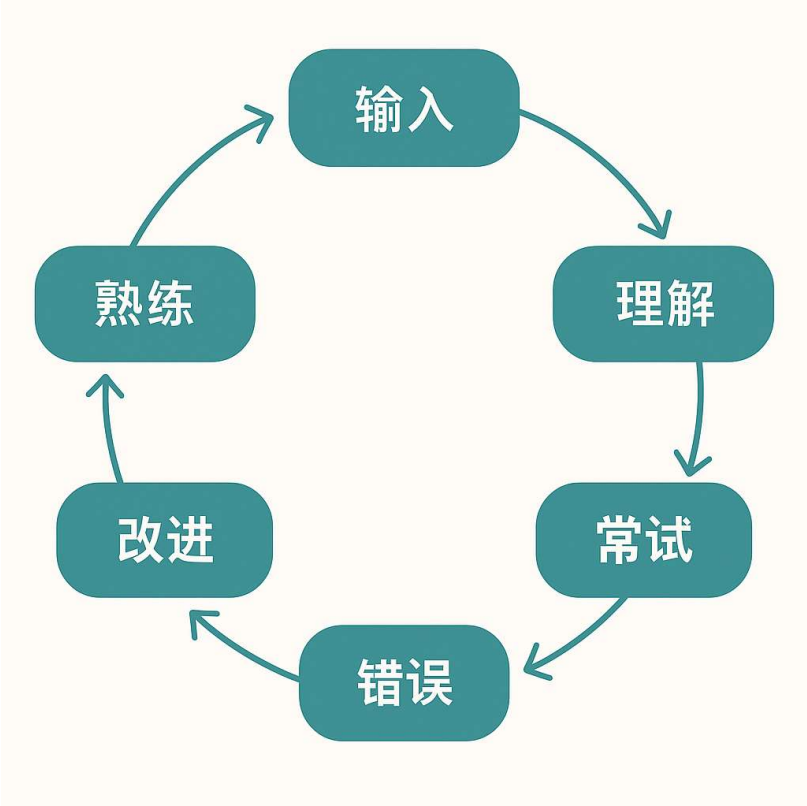
改进：调整理解或方法

根据反馈，有时候需要做出修改和变通。不仅要把代码改对，而且要改得更好，更优。这是一种工匠精神，如果缺乏这种精神，你可能不但难以成为优秀的程序员，甚至可能连入门都困难。因为，改进是一种主体态度，缺乏这种主体的思考，你将陷入浅尝辄止，骄傲自满的境地，自然不可能学好编程。

后续的课程包括代码重构，就是针对如何改进代码的。

熟练：技能内化，形成能力

熟练是上述行为的自然结果，但是，熟练不是终点，而是新的开始。通过前述的学习，能够掌握一部分技能，但是，也会发现新的需要强化的地方，这就要求你去探索新的侧面，新的方法，新的领域。必将引导你进入一轮新的认知循环。



插图：圆环型“学习闭环图”，七个阶段

1.3 编程学习的特殊性

◆ 内容的多样性

编程学习是高度综合的过程，不仅要学会编代码，还要学会操作编程工具，还要熟悉程序的运行环境。

代码是对现实社会的反映，这有点像作家，如果缺乏对社会的认识就会变成空中楼阁。

◆ 实践性

与语文、历史不同，它是“构造型”的技能，要不断输出产物（代码）。要不断试错，学习中“错误-反馈-改进”的环节特别频繁，是一种**强反馈型学习**

学习的 80%实践在实践，20 的时间在输入知识。

◆ 知识体系庞大

编程需要的知识体系非常庞大，从各种各样的编程语言之外需要网络知识，数据库，计算机原理等。这些还是基础部分，随着技术的深入还要熟悉软件工程的各个环节的知识。

不过，不要急，慢慢来，这些知识是有体系的，先从基础部分开始吧。。。

◆ 前沿性

编程是一个前沿的邻域，这些知识可以说是日新月异，每天都有新的技术出现，技术迭代特别快，需要我们不断更新自己的知识体系，也就是需要持续学习。

1.4 为什么很多人“学不好编程”？

◆ 常见误区：

把编程当作死记硬背（停留在“学”的阶段而忽视了“习”的过程）

害怕出错，回避“尝试”与“反馈”

忽略环境与工具的学习，直接扑在代码细节上

实质上是没有建立起编程学习的“完整认知闭环”

大家反思一下，有没有进入以上的误区？ 【点名问】

1.5 编程学习的闭环实践建议

输入：看教材、教程、视频、样例代码

理解：用自己的话复述，画流程图，讲给别人听

尝试：从模仿代码开始，到独立写小功能

错误：观察程序出错的提示，不惧怕失败，错误信息有点类似于一个人生病的病症，如发烧，咳嗽等。医生证实更具这些症状来诊断病人，从而达到治病的目的。所以，要积极看待错误信息。

说说 你对错误信息的反应方式？

反馈：利用调试器、日志、同伴讨论、AI 工具

改进：总结问题、更新方法、记录调试日志

熟练：不断练习，形成编程直觉

小练习建议：写一个“加法函数”，然后故意制造两个 Bug，让学生自己找错、调试

请调试下面的代码：

```
public double add ( double a, double b) {
    long result = a + b;
    return result;
}
```

第二部分：编程学习的三个层面

2.1 掌握环境：搭建运行场

操作系统：Windows（或者 MacOS）

安装 IDE、解释器（如 Java + Eclipse）

相信大家都已经有所了解，这里就不一一重复了。

大家可以沿用自己最中意，最熟悉的运行环境。

但是，最好熟练掌握一种，同时扩大视野，知道编程环境的多样性。

各种环境既有共同之处，也有各自的特点，要能够在变通的环境下编程。

不同的语言，不同的操作系统，不同的 IDE，不同的依赖库

都要精通一种，兼顾多种。

类比：程序 = 种子；环境 = 土壤。无环境，程序无法“发芽”。

2.2 掌握工具：提高效率的装备

编辑器（如 Eclipse, IDEA, VS Code）

命令行工具（终端、PowerShell）

调试器（断点、堆栈）

版本控制（Git + GitHub）

AI 助手：搜索、ChatGPT、Copilot

2.3 掌握代码：学习表达思维

基础语法（变量、流程控制、函数）

数据结构与抽象（数组、类、对象）

编程语言的本质：人与机器的交流语言

类比：代码是“乐谱”，计算机是“演奏者”，程序员是“作曲家”。

✅ 小互动：

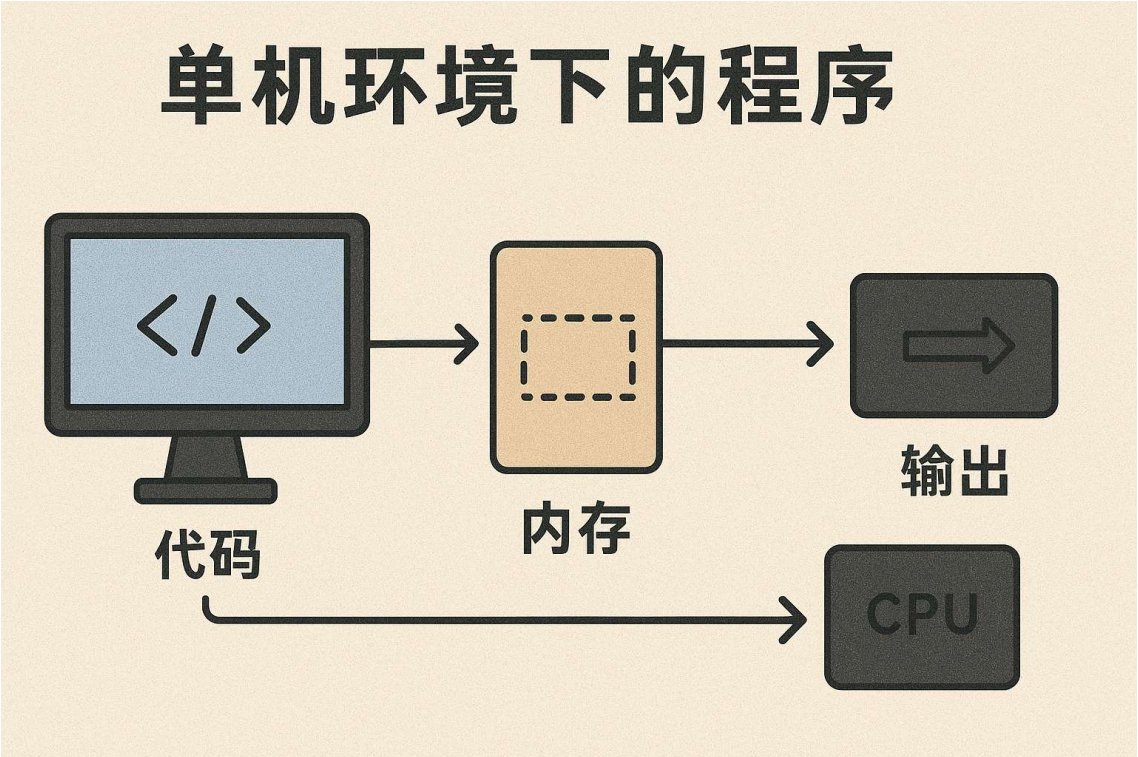
请学生想象“写作业”时用的工具，映射到编程工具

第三部分：程序如何被执行？

3.1 单机环境下的程序

编写代码 → 编译 → 加载到内存 → JVM 执行 → 输出结果

示例：Java 代码运行流程



*（符合冯诺依曼体系结构的单机程序运行模型图）

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

提问：

1）这个小程序里的输出是什么？ 输出设备是什么？

2）这个小代码里都有什么关键字（Keyword）？

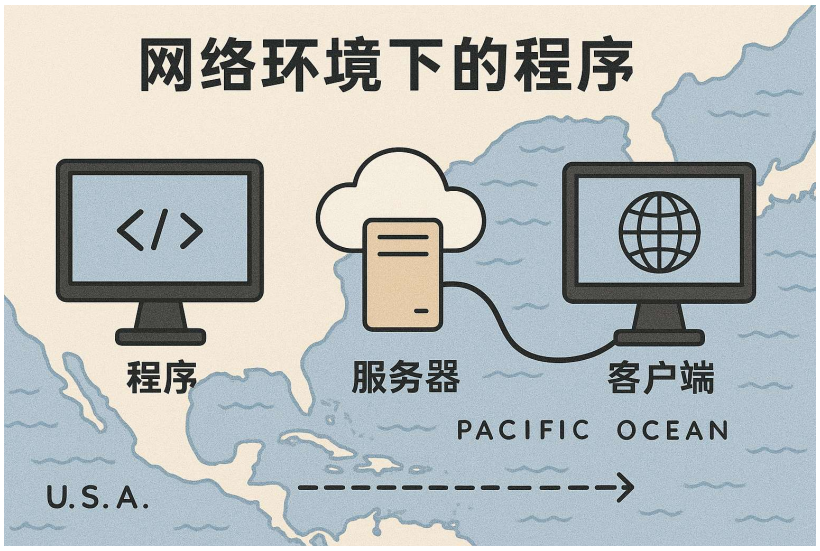
3.2 网络环境下的程序

客户端/服务器结构

浏览器发送请求 → 服务器返回页面

后端语言运行位置，数据库、API 等协作角色

✅ 实操：使用 Java + Eclipse 构建简单 HTTP 服务器（或使用 Spring Boot 示例）



网络环境下的程序实行是复数计算机资源的协调运行，包括客户端，网络，服务器群。

这些计算机资源在地理上是分散的，通过网络和数据传输协议进行协作。

所以，网络环境下的程序运行要比单机环境下的复杂 10 倍以上，你甚至可以认为整个互联网就是一台巨大的计算机，你的代码有可能跨越整个互联网在运行。

网络编程要求程序员有广阔的视野，理解复杂模型的能力。

这样，我们有可能对网络程序有高不可攀的感觉。

为了让大家对网络程序去魅，下面让我们来调试一个最简单的网络程序，它是所有网络程序的原型，这里是你通往网络编程的起点，更加复杂的网络程序只不过同样的技术不断组合，衍生的结果而已，用不着害怕。

这个小程序是一个简单的 Web APP，可以通过浏览器进行访问，启动。

希望你们通过调试这个程序破除对程序的神秘感，从而更加自信地进入下一阶段学习！

```
/**
 *
 */
package http;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class SimpleHttpServer {

    private static final int PORT = 8088;
    private static final int THREAD_POOL_SIZE = 10;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("HTTP Server started on port " + PORT);
            System.out.println("Endpoints:");
            System.out.println("  http://localhost:" + PORT + "/");
            System.out.println("  http://localhost:" + PORT + "/hello");

            // 创建线程池处理并发请求
```

```
        ExecutorService executor = Executors.newFixedThreadPool(THREAD_POOL_SIZE);

        while (true) {
            Socket clientSocket = serverSocket.accept();
            executor.submit(() -> handleRequest(clientSocket));
        }
    } catch (IOException e) {
        System.err.println("Server error: " + e.getMessage());
    }
}

private static void handleRequest(Socket clientSocket) {
    try (BufferedReader in = new BufferedReader(
            new InputStreamReader(clientSocket.getInputStream()));
         OutputStream out = clientSocket.getOutputStream()) {

        // 读取请求的第一行
        String requestLine = in.readLine();
        if (requestLine == null) return;

        // 解析请求路径
        String path = parsePath(requestLine);

        // 根据路径生成响应
        byte[] response;
        String contentType;

        if ("/hello".equals(path)) {
```

```
                response = "{\"message\": \"Hello, World!\"}".getBytes(StandardCharsets.UTF_8);
                contentType = "application/json";
            } else {
                String welcome = "<html><body><h1 style='color:red'>Welcome to Java HTTP Server!</h1>"
                        + "Available endpoints:\n"
                        + "  /hello - Returns JSON greeting</body></html>";
                response = welcome.getBytes(StandardCharsets.UTF_8);
                contentType = "text/html";
            }

            // 发送 HTTP 响应
            sendResponse(out, 200, "OK", contentType, response);

        } catch (IOException e) {
            System.err.println("Request handling error: " + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
                System.err.println("Error closing client socket: " + e.getMessage());
            }
        }
    }

    private static String parsePath(String requestLine) {
        String[] parts = requestLine.split(" ");
        if (parts.length >= 2) {
            return parts[1]; // 返回请求路径
```

```
        }
        return "/";
    }

    private static void sendResponse(OutputStream out, int statusCode, String statusText,
                                     String contentType, byte[] content) throws IOException {
        // 构建 HTTP 响应头
        String header = "HTTP/1.1 " + statusCode + " " + statusText + "\r\n" +
                        "Content-Type: " + contentType + "; charset=utf-8\r\n" +
                        "Content-Length: " + content.length + "\r\n" +
                        "Connection: close\r\n" +
                        "\r\n";

        // 发送响应头
        out.write(header.getBytes(StandardCharsets.UTF_8));
        // 发送响应体
        out.write(content);
        out.flush();
    }
}
```

在 Eclipse 或者 IDEA 环境下建一个工程，并且实行它

浏览器访问 http://localhost:8088

四、课程结语与引导

真正的学习是“学”+“习”构成闭环

编程的学习不是跳入代码，而是逐步掌握“环境➡工具➡代码”

在以后的学习过程中，我们可能面对很多困难，我希望你们掌握以下的能力

如何应对编程学习焦虑

如何问问题

如何解读代码

如何调试代码

如何整理思维过程

如何解读仕样和设计书

如何做设计

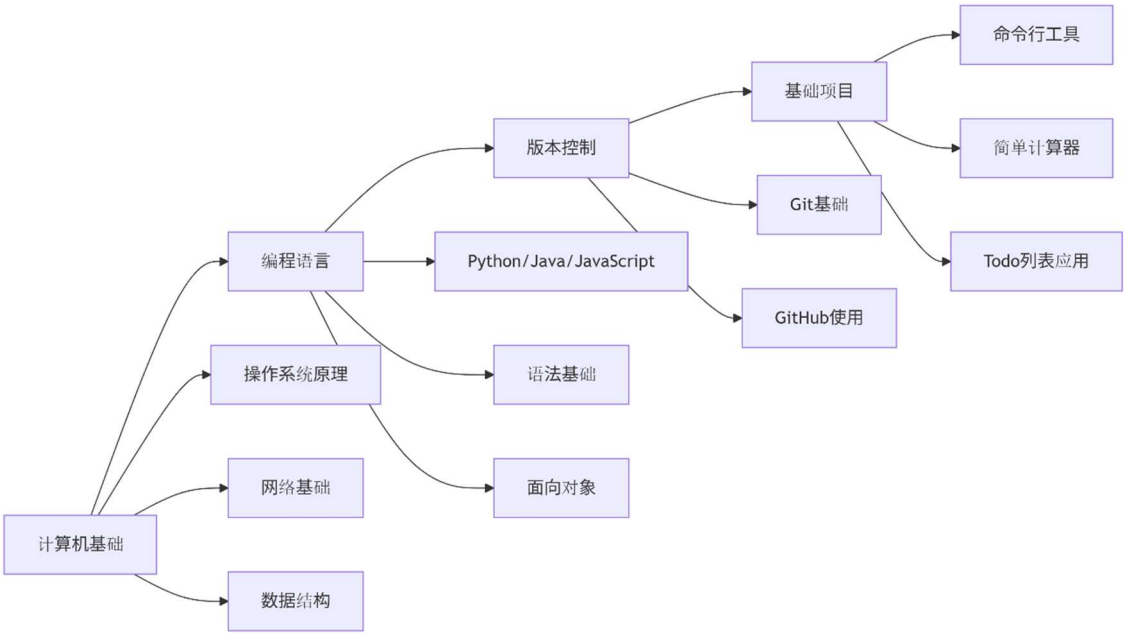
如何去认知一个概念

如何建立思维模型

将在以后的课程中逐一讲解

五、附加材料：

编程学习地图（心智图）



软件推荐：Java, Eclipse, Git, Terminal

视频资源推荐：《计算机是如何运行的》（推荐 B 站视频）

AI 辅助编程演示（如 GitHub Copilot）

◆ 写作和编程的异同

不瞒大家，我写这个资料使用了 ChatGPT 和 DeepSeek 的辅助。发现写作其实也很难，即便用了 AI 的功能。大家在学习编程的时候，我发现，我需要学习写作。否则，写出来的东西会对不起大家。

我也惊奇的发现，写作和编程有惊人的相似之处，我发现，我在告诫你们如何学习编程的那些元认知，正好也可以用来指导我去把资料写好。反过来，写作，对于大家相对比较容易想象，所以我把从 DeepSeek 上获得的，关于写作和编程的异同点也编排进去，大家可以通过想象写作，来反思编程，也能获得一定的启发。

编程，和写作的相似处大家可以参考，但是，编程和写作有非常不同。

大家要注意别把写作可以用的对于比如类比，引申，比喻中的各种模糊论述的容忍复制到编程里去，因为，程序是要非常严谨，不能有任何歧义的。通过比较写作和编程的异同之处，既要把写作可以用的手法应用到编程，又要把程序的严密性放到首位，这样才能写出坚牢的代码。

问： 编程和写作有什么异同点？

DeepSeek：

好的，写作和编程虽然一个通常被视为人文艺术，一个被视为科学技术，但它们在核心上都是**创造性的表达和沟通过程**，旨在将想法转化为特定的形式。它们之间存在着有趣的异同点：

核心相似点

1. **都是创造性活动：**

* **写作：** 从无到有地构建故事、论点、描述或情感。

* **编程：** 从无到有地设计算法、构建功能、解决特定问题。

* *都需要想象力、构思和将抽象想法具象化的能力。*

2. **都需要结构化思维：**

* **写作：** 需要清晰的逻辑结构（如引言、正文、结论；起承转合）、段落组织、句子流畅度。好的文章有内在的“架构”。

* **编程：** 核心就是结构化和逻辑。需要设计代码结构（模块、函数、类）、控制流程（条件、循环）、数据组织。程序的架构至关重要。

* *都需要规划和组织思想/指令，使其连贯、有效。*

3. **都强调清晰和精确的表达：**

* **写作：** 追求用词准确、避免歧义、语法正确，以便读者准确理解意图。

* **编程：** 要求语法绝对精确（一个分号都不能错）、命名清晰、逻辑无歧义，以便计算机能准确执行且其他程序员能理解。

* *模糊不清的表达在写作中会造成误解，在编程中会导致错误或崩溃。*

4. **都需要考虑受众/用户：**

* **写作：** 作者必须考虑读者的知识水平、兴趣和需求，调整语言、风格和内容。

* **编程：** 程序员必须考虑最终用户（用户界面、用户体验）和协作的程序员（代码可读性、可维护性）。API 设计就是为用户（其他开发者）服务的。

* *成功的作品/程序都离不开对接收者的理解。*

5. **都涉及迭代和修改：**

    * **写作：** 草稿、修改、润色、编辑是标准流程。很少有“一次成稿”的完美作品。

    * **编程：** 编写、测试、调试、重构、优化是开发的核心循环。代码需要不断迭代完善。

    * *两者都认识到初稿/初版很少是完美的，需要反复打磨。*

6. **都使用特定的规则和约定：**

    * **写作：** 遵循语法、拼写、标点、体裁（如小说、议论文、诗歌）的规则和风格指南。

    * **编程：** 遵循编程语言的语法、库的 API 规范、设计模式、编码规范。

    * *掌握这些规则是有效表达的基础。*

核心不同点

1. **最终目标和产物：**

    * **写作：** 目标通常是**传达信息、讲述故事、说服、表达情感或引发思考**。产物是**供人类阅读和理解的文本**（文章、小说、诗歌、报告等）。

    * **编程：** 目标是**指示计算机执行特定任务、解决问题或实现功能**。产物是**供计算机执行的指令序列（代码）**，其最终输出可能是一个软件、网站、工具或数据处理结果（供人或机器使用）。

2. **执行者和执行环境：**

* **写作：** 由**人类读者**解读和执行（理解、感受、行动）。解读依赖于读者的主观经验、知识和背景。

* **编程：** 由**计算机**（编译器/解释器）执行。执行是客观、字面、精确的，严格按照代码指令进行，没有主观解读的空间。

3. **灵活性与模糊性容忍度：**

* **写作：** 相对**灵活**。允许甚至鼓励模糊性、隐喻、多义性、风格化表达。语法规则有时可以为了艺术效果而打破（诗歌尤甚）。人类读者能处理一定程度的不精确和上下文推断。

* **编程：** **极其严格**。**几乎不容忍任何模糊性**。语法必须100%正确，逻辑必须完整且无矛盾。计算机不会“猜”你的意图，一点小错误（如拼写错误、漏掉括号）都会导致失败（编译错误、运行时错误、逻辑错误）。

4. **逻辑的严谨性要求：**

* **写作：** 需要逻辑，但逻辑链条可以隐含、跳跃，或服务于情感和艺术表达。论证需要逻辑，但叙述可以是非线性的。

* **编程：** **逻辑的严谨性是绝对核心**。程序的行为完全由其代码逻辑决定。所有可能性（分支条件、边界情况）都需要明确考虑和处理。逻辑错误是程序Bug的主要来源之一。

5. **工具和媒介：**

* **写作：** 主要工具是语言（词汇、语法、修辞）。媒介是文本。

* **编程：** 工具是编程语言（语法、语义、标准库）、开发环境（IDE）、调试器、版本控制系统等。媒介是代码文本，但最终作用于计算机硬件/软件环境。

6. **反馈机制：**

* **写作：** 反馈来自读者（评论、讨论、销量、个人感受），通常是主观的、延迟的、定性的（“这段写得真好/不好”）。

* **编程：** 反馈来自计算机（编译错误、运行时错误、测试失败/通过）和用户（Bug报告、使用反馈），以及性能指标（速度、资源消耗）。反馈通常是客观的、即时的（编译时）、可量化的（测试覆盖率、响应时间）。

7. **“正确性”的定义：**

* **写作：** “正确性”相对主观，可能包括语法正确、事实准确、论证有力、情感真挚、文笔优美、符合体裁要求等。没有唯一标准答案。

* **编程：** “正确性”有更客观的标准：程序是否能完成既定功能？是否满足需求规格？是否无Bug？是否高效？结果是否可预期和可重复？通常有明确的测试用例来验证。

总结

* **同：** 两者都是**创造性、结构化、需要清晰表达、考虑受众、迭代进行、遵循规则**的智力活动。

* **异：** 根本区别在于**目标和执行者**。写作是**面向人类**的**思想/情感表达**，**容忍模糊和风格**；编程是**面向计算机**的**精确指令**，**要求绝对逻辑严谨和无歧义**。

可以形象地说：

* **写作是“人”与“人”之间的沟通艺术。**

* **编程是“人”与“机器”之间的沟通科学（也包含艺术性，尤其在架构和设计层面）。**

理解这些异同点，有助于写作者借鉴编程的结构化和逻辑性，也帮助程序员认识到清晰文档和用户沟通（这也是一种写作）的重要性。两者技能的融合（如技术写作、清晰的代码注释、用户友好的文档）在现代社会越来越有价值。

我在后续的课程里会讲到，如何做到《从模糊到确信》，用来排除编程中的模糊之处，写出正确的没有歧义和坚牢的设计书及代码。