高等程序设计 - Qt/C++

第 1 章：开发环境搭建

王培杰

长江大学地球物理与石油资源学院

2025 年 9 月 6 日

课程简介

- 本课程系统介绍 C++ 语言基础与 Qt 应用开发，涵盖从环境搭建到高级编程范式与项目实践。
- 通过理论讲解与大量代码示例，帮助学生掌握现代 C++ 编程思想和跨平台 GUI 开发能力。
- 课程内容包括：C++ 语法、面向对象、模板与泛型、Qt 框架、信号与槽、设计模式等。
- 强调实践操作，配套完整的工程模板和实验指导，助力学生独立开发实际应用。
- 适合有一定编程基础、希望深入学习 C++ 和 Qt 开发的相关专业学生。

目录

1. 编程概念入门
2. 软件开发概述
3. Qt 开发环境安装
4. Qt Creator IDE
5. 编译器和调试配置
6. 版本控制集成
7. 项目实践
8. 其他 IDE 配置

什么是编程?

编程的本质

编程就像是给计算机下达指令，让它按照我们的想法去工作和解决问题。

为什么要学习编程?

- **解决问题**：编程可以帮助我们自动化处理重复性工作，提高效率
- **创造价值**：通过编程创造新的工具、应用和服务
- **思维训练**：编程培养逻辑思维、问题分解和系统化思考能力
- **就业机会**：编程技能在现代社会越来越重要，就业前景广阔

编程语言的层次

机器语言（第一代语言）

- 计算机能直接理解的二进制代码（0 和 1）
- 例如：10110000 01100001
- 优点：执行速度最快
- 缺点：编写困难，难以理解和维护

汇编语言（第二代语言）

- 使用助记符表示机器指令
- 例如：MOV AX, 61H
- 优点：接近机器语言，执行效率高
- 缺点：仍然复杂，依赖特定硬件

高级语言的特点

- 使用接近自然语言的语法
- 例如：C++, Java, Python 等
- 优点：易学易用，可读性强，可移植性好
- 缺点：需要编译或解释，执行效率相对较低

编译型 vs 解释型

编译型语言

- C++, Java, C#
- 先编译后执行
- 执行速度快

解释型语言

- Python, JavaScript
- 边解释边执行
- 开发速度快

编程思维的培养

计算思维的核心要素

- **分解问题**：将复杂问题分解为简单的小问题
- **模式识别**：发现问题中的规律和重复模式
- **抽象思维**：提取问题的本质，忽略不重要的细节
- **算法思维**：设计解决问题的步骤和方法

编程思维的实际应用

- 日常生活中的问题解决
- 工作流程的优化
- 系统化思考能力的提升

程序是什么?

程序的本质

程序是一系列指令的有序集合，告诉计算机应该做什么、怎么做。

程序的生命周期

1. **编写源代码**：使用编程语言编写程序
2. **编译**：将源代码转换为机器能理解的形式
3. **测试**：验证程序是否按预期工作
4. **部署**：将程序发布给最终用户使用
5. **维护**：根据用户反馈和需求变化进行更新

程序是如何工作的?

程序执行的过程

- **源代码编写**：程序员使用高级编程语言（如 C++）编写源代码
- **编译**：编译器将源代码翻译为机器语言
- **链接**：链接器将多个目标文件和库文件合并
- **加载**：操作系统将程序加载到内存中
- **执行**：CPU 逐条执行指令，实现程序功能

关键概念

- **编译器**：将高级语言转换为机器语言的工具
- **链接器**：将多个目标文件合并为可执行文件的工具
- **可执行文件**：计算机可以直接运行的程序

程序的基本组成

程序的三要素：IPO（Input、Process、Output）模型

1. **输入（Input）**：获取外部数据
   - 鼠标、键盘、触摸屏输入
   - 文件读取
   - 网络数据
   - 传感器数据
2. **处理（Process）**：对数据进行操作
   - 数学计算
   - 逻辑判断
   - 数据转换
   - 信息处理
3. **输出（Output）**：展示处理结果
   - 屏幕显示
   - 文件保存
   - 打印输出
   - 网络发送

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

软件开发概述

- **软件开发**：使用编程语言和开发工具，按照一定的流程和规范，开发出满足用户需求的软件产品的过程。
- **软件开发流程**：软件开发过程中的一系列步骤和活动，包括**需求分析、设计、编码、测试、部署和维护**等。
- **软件开发工具**：软件开发过程中使用的工具，包括**编译器、调试器、集成开发环境（IDE）**等。
- **软件开发语言**：软件开发过程中使用的语言，包括 **C、C++、Java、Python** 等。
- **软件开发框架**：软件开发过程中使用的框架，包括 **Qt、MFC、DotNet、Spring、Vue** 等。

开发环境简介

定义

开发环境是程序员用于编写、调试和运行代码的一整套软件工具集合。正如木匠需要工具箱，程序员也需要合适的开发环境来高效完成工作。

重要性

- **降低学习门槛**：自动化配置，初学者易上手
- **提升效率**：专注编程逻辑，自动编译、调试、部署
- **减少错误**：标准化环境，避免”只在我电脑上能运行”
- **养成规范**：使用专业工具，统一团队开发标准
- **便于协作与管理**：集成版本控制和自动化构建，方便项目维护

开发环境的组成与流程

主要组成

- **编译器**：如 GCC、Clang、MSVC、MinGW、Intel C++ 等，将源代码转为可执行程序
- **框架**：提供图形界面和跨平台支持，提供丰富的组件和工具，如 Qt、MFC、DotNet、Spring、Vue 等
- **集成开发环境（IDE）**：代码编辑、调试、项目管理，如 Qt Creator、Visual Studio、Eclipse、VS Code 等
- **调试工具**：定位和修复程序错误，如 GDB、LLDB、MSVC 等
- **版本控制系统**：管理代码历史与团队协作，如 Git、SVN、TFS 等

典型工作流程

1. **编写代码**：
   在 IDE 中进行代码编写
2. **编译代码**：
   通过编译器生成可执行文件
3. **调试程序**：
   利用调试器查找和修复问题
4. **运行程序**：
   执行编译后的应用
5. **版本控制**：
   保存和管理代码变更

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

什么是 Qt?

Qt 简介

Qt 是一个跨平台的 C++ 应用程序开发框架，由 Qt Company 开发。它提供了丰富的图形界面组件和开发工具。

Qt 的优势

- **跨平台**：一套代码可在 Windows、 macOS、 Linux 运行
- **易学易用**：提供直观的图形界面设计工具
- **功能丰富**：包含 GUI、网络、数据库、多媒体等模块
- **文档完善**：有详细的中文文档和示例
- **社区活跃**：大量学习资源和社区支持
- **免费使用**：开源免费，无需付费许可证
- **基于 C++**：C++ 是 Qt 的官方语言，使用 C++ 开发 Qt 应用程序，可以充分发挥 C++ 的性能和灵活性

Qt 安装方式

在线安装器（推荐）

- **Qt 官方推荐**：最标准的安装方式
- **自动管理依赖**：自动下载所需组件
- **可选择组件**：只安装需要的模块
- **支持离线安装**：下载后可离线使用
- **免费账户**：注册 Qt 账户即可使用

离线安装包（不推荐）

- **完整安装包**：包含所有组件
- **无需网络连接**：适合网络受限环境
- **适合企业环境**：便于批量部署
- **文件较大**：通常几 GB 大小
- **需要许可证**：商业使用需要付费

源码编译安装（太复杂、不推荐）

- **下载源码**：从 Qt 官方网站下载源码
- **编译源码**：使用编译器编译源码
- **安装源码**：使用安装器安装源码

安装前的准备工作

系统要求

- **操作系统**：Windows 10/11、macOS 10.14+、Ubuntu 18.04+ 等
- **硬件配置**：至少 4GB 内存，推荐 8GB 以上
- **磁盘空间**：至少 20GB 可用空间
- **网络连接**：稳定的网络连接（在线安装）

需要准备的内容

- **Qt 账户**：在 qt.io 注册免费账户（推荐）
- **邮箱地址**：用于注册和接收通知
- **耐心**：首次安装可能需要较长时间

Qt 安装步骤

1. **下载 Qt 在线安装器**
   - 访问 `https://www.qt.io/download`
   - 选择”Community User”
   - 选择”Download Qt Online Installer”
   - 下载适合您操作系统的版本（推荐 Windows 10/11）
   - 下载后运行安装器
2. **注册 Qt 账户（免费）**
   - 使用邮箱注册免费账户
   - 验证邮箱地址
   - 登录安装器
3. **选择安装路径**
   - 建议使用默认路径（推荐 C:\Qt）
   - 足够的磁盘空间（推荐 20GB 以上）
   - 避免中文路径
4. **选择 Qt 版本和组件**
   - 选择最新的 LTS 版本
   - 选择 MinGW 和/或 MSVC 编译器
   - 选择 Qt Creator IDE
5. **配置编译器**
   - 安装器会自动配置
   - 无需手动设置
   - 等待安装完成
6. **完成安装**
   - 验证安装是否成功
   - 运行 Qt Creator 测试
   - 创建第一个项目

推荐配置与安装时间

推荐配置

- **Qt 6.9.1 LTS 版本**：长期支持，稳定可靠
- **MinGW+MSVC 编译器**：Windows 下推荐
- **Qt Creator IDE**：官方集成开发环境
- **Qt Designer**：可视化界面设计工具
- **Qt Debugging Tools**：调试工具集
- **Qt Creator 17.0.0 (community)**：官方集成开发环境
- **Qt Modules**：Qt 模块，如 Qt Charts、Qt Network、Qt PDF、Qt XML、Qt SVG、Qt WebView 等

安装常见问题及解决方案

网络问题

- **下载慢**：用国内镜像或 VPN
- **中断、超时**：用下载器，支持断点续传，检查网络，重试

安装问题

- **权限不足**：用管理员身份运行，检查权限
- **组件缺失**：检查安装选项，重新安装缺失组件
- **安装失败**：查系统要求，重装，检查杀毒软件

配置问题

- **编译器未找到**：重装或手动配置，检查编译器路径
- **项目无法编译**：查 Qt 版本与编译器，检查编译器路径

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

什么是 Qt Creator?

Qt Creator 简介

Qt Creator 是 Qt 官方提供的集成开发环境（IDE），专门为 Qt 开发而设计。它集成了代码编辑、调试、界面设计等功能于一体。

为什么选择 Qt Creator?

- **官方支持**：Qt 公司官方开发，与 Qt 框架完美集成
- **免费使用**：开源免费，无需付费许可证
- **功能完整**：包含开发所需的所有工具
- **易于学习**：界面友好，适合初学者
- **跨平台**：支持 Windows、macOS、Linux

Qt Creator 特性

核心功能

- **智能代码补全**：自动提示函数、变量、类名
- **语法高亮**：不同颜色显示代码结构
- **实时错误检查**：编写时即时发现语法错误
- **集成调试器**：内置调试工具，支持断点
- **可视化设计器**：拖拽式界面设计
- **项目管理**：统一管理项目文件和配置

界面布局

- **编辑器区域**：主要代码编辑区域
- **项目导航**：文件树和项目结构
- **输出窗口**：编译输出和错误信息
- **调试控制台**：调试信息和变量查看

Qt Creator 界面详解

主要区域

- **菜单栏**：文件、编辑、构建、调试等菜单
- **工具栏**：常用功能的快捷按钮
- **模式选择器**：编辑、设计、调试等模式
- **项目导航**：文件树和项目结构
- **编辑器**：代码编辑主区域
- **输出窗口**：编译输出和错误信息

侧边栏面板

- **项目**：项目文件和结构
- **打开文档**：当前打开的文件
- **书签**：代码书签管理
- **文件系统**：本地文件浏览
- **类视图**：类和方法导航
- **大纲**：当前文件结构

Qt Creator 界面配置

主题设置

- **浅色主题（Light）**：适合白天使用，减少眼疲劳
- **深色主题（Dark）**：适合夜间使用，保护眼睛
- **高对比度主题**：适合视力不佳的用户
- **自定义颜色方案**：根据个人喜好调整

编辑器设置

- **字体和字号**：选择适合的编程字体（如 Consolas、 Source Code Pro）
- **缩进设置**：设置 Tab 键和空格键的行为
- **代码折叠**：隐藏不需要查看的代码块
- **行号显示**：显示代码行号，便于定位

Qt Creator 使用技巧

快捷键（必学）

- **Ctrl+Space**：代码补全
- **F5**：开始调试
- **Ctrl+R**：运行程序
- **Ctrl+B**：构建项目
- **F9**：设置/取消断点
- **Ctrl+F**：查找文本
- **Ctrl+Shift+F**：全局查找
- **Ctrl+/**：注释/取消注释

初学者建议

- **熟悉界面**：先了解各个区域的功能
- **使用代码补全**：提高编程效率
- **学会调试**：掌握基本的调试技巧
- **查看帮助**：善用 F1 键查看文档
- **保存项目**：定期保存，避免丢失

什么是 IDE?

IDE 的定义

集成开发环境（Integrated Development Environment，IDE）是一种软件应用程序，为程序员提供全面的开发工具，包括代码编辑器、编译器、调试器、项目管理等功能的统一界面。

IDE vs 文本编辑器

- **IDE**：功能全面，适合大型项目开发
- **文本编辑器**：轻量级，适合简单脚本编写
- **IDE 优势**：代码补全、语法检查、集成调试
- **编辑器优势**：启动快速、占用资源少

主流 IDE 介绍

Qt Creator

- **专为 Qt 设计**：与 Qt 框架完美集成
- **免费开源**：无需付费许可证
- **跨平台**：Windows、macOS、Linux
- **可视化设计**：内置 Qt Designer
- **适合初学者**：界面友好，易上手

Visual Studio

- **微软官方**：Windows 平台最强大 IDE
- **功能丰富**：企业级开发工具
- **社区版免费**：个人和小团队使用
- **调试强大**：业界最佳调试体验
- **扩展丰富**：大量插件和扩展

VS Code

- **轻量级**：启动快速，占用资源少
- **免费开源**：微软开发，完全免费
- **扩展生态**：丰富的插件市场
- **跨平台**：支持所有主流操作系统
- **现代化**：界面美观，用户体验佳

IDE 选择建议

初学者推荐

- **首选 Qt Creator**：专为 Qt 开发设计，学习成本低
- **界面友好**：可视化设计工具，直观易懂
- **文档完善**：官方文档和教程丰富
- **社区支持**：Qt 社区活跃，问题容易解决

进阶用户选择

- **Visual Studio**：大型项目，企业级开发
- **VS Code**：轻量级开发，多语言支持
- **Cursor**：AI 辅助开发，提高效率

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

什么是编译器?

编译器的作用

编译器是一种将人类编写的源代码转换为计算机能够理解和执行的机器语言的工具。它的作用类似于”翻译官”，将高级语言翻译成低级语言，使计算机能够正确执行我们的程序。

编译器的分类

- **本地编译器**：生成当前平台可执行代码，如 GCC、MSVC
- **交叉编译器**：生成其他平台可执行代码，如 Android NDK
- **即时编译器 (JIT)**：运行时动态编译，如 Java JVM、.NET CLR

GNU 编译器套件（GCC）简介

什么是 GCC?

GNU 编译器套件（GNU Compiler Collection，简称 GCC）是由 GNU 项目开发的自由、开源的编译器集合。最初仅支持 C 语言，现已支持 C++、Objective-C、Fortran、Ada、Go、D 等多种编程语言。GCC 是 Linux、Unix 等类 Unix 系统的默认编译器，也是开源软件开发的核心工具之一。

GCC 的主要特点

- **跨平台**：支持多种操作系统（Linux、Windows、macOS 等）和多种硬件架构（x86、ARM、RISC-V 等）。
- **多语言支持**：不仅支持 C/C++，还支持 Fortran、Ada、Go 等多种语言。
- **优化能力强**：内置多种优化选项，可生成高效的目标代码。
- **开源免费**：遵循 GPL 协议，源代码开放，社区活跃。
- **工具链完善**：包含编译器（gcc/g++）、汇编器（as）、链接器（ld）、调试信息生成等工具。

GCC 编译指令

GCC 常用命令

- `gcc`：C 语言编译器
- `g++`：C++ 语言编译器
- `gfortran`：Fortran 编译器
- `as`：汇编器
- `ld`：链接器

GCC 编译流程示例

- `gcc hello.c -o hello` // 编译 C 程序
- `g++ main.cpp -o main` // 编译 C++ 程序
- `gcc -E hello.c -o hello.i` // 仅预处理
- `gcc -S hello.c -o hello.s` // 生成汇编代码
- `gcc -c hello.c -o hello.o` // 生成目标文件

MinGW 简介

什么是 MinGW?

MinGW（Minimalist GNU for Windows）是一个在 Windows 平台上使用的轻量级 GNU 开发环境。它为 Windows 系统提供了 GCC（GNU Compiler Collection）编译器及相关工具，使开发者能够在 Windows 下编译和构建 C、C++ 等语言的本地应用程序。

MinGW 的主要特点

- **原生编译**：生成可在 Windows 上直接运行的本地可执行文件（.exe），无需额外的运行时环境。
- **开源免费**：遵循 GPL 协议，完全免费，适合个人和商业用途。
- **兼容性好**：支持标准 C/C++，可与大多数开源库和工具链配合使用。
- **体积小巧**：安装包体积较小，安装和配置简单。
- **包含常用工具**：集成 GCC、G++、GDB、Make 等开发工具。

MSVC 简介

什么是 MSVC?

MSVC（Microsoft Visual C++）是微软公司开发的 C/C++ 编译器和集成开发环境（IDE）的一部分，广泛应用于 Windows 平台的本地应用程序开发。MSVC 不仅提供了强大的编译、调试和优化功能，还集成了丰富的开发工具和库，支持现代 C++ 标准。

MSVC 的主要特点

- **深度集成 Windows**：与 Windows 操作系统和 API 高度兼容，适合开发各类 Windows 桌面、服务和驱动程序。
- **强大的 IDE 支持**：配合 Visual Studio，拥有智能补全、代码重构、图形化调试、性能分析等高级功能。
- **标准兼容性好**：持续跟进 C++ 标准，支持 C++11/14/17/20 等现代特性。
- **优化能力强**：内置多种优化选项，生成高效的本地代码，适合对性能有较高要求的项目。
- **丰富的库支持**：集成 Microsoft STL、MFC、ATL 等丰富的开发库，便于快速开发大型应用。

MSVC 编译指令

常用 MSVC 命令行工具

- `cl.exe`：C/C++ 编译器
- `link.exe`：链接器
- `nmake.exe`：微软的 make 工具
- `devenv.exe`：Visual Studio IDE 启动器

MSVC 编译示例

- `cl main.cpp` // 编译 C++ 源文件，生成可执行文件 `main.exe`
- `cl /LD mylib.cpp` // 编译为动态链接库（DLL），生成 `mylib.dll`
- `cl /c foo.cpp bar.cpp` // 仅编译为目标文件（.obj），不进行链接
- `link foo.obj bar.obj /OUT:app.exe` // 手动链接多个目标文件，生成可执行文件
- `cl /I include_path main.cpp` // 指定头文件搜索路径

编译过程-预处理

什么是预处理?

预处理是 C/C++ 编译过程的第一步，主要由预处理器完成。它在正式编译前对源代码进行一系列文本处理，生成" 预处理后的代码"，为后续编译、汇编和链接做好准备。

预处理的主要功能

- 宏替换（#define）
- 条件编译（#ifdef/#endif）
- 头文件包含（#include）
- 特殊指令处理（如 #pragma）

编译过程-预处理示例

预处理常用指令及详细示例

```
// 预处理指令示例
#define MAX 100                 // 定义常量宏，将 MAX 替换为 100
#define SQUARE(x) ((x)*(x))     // 带参数宏，SQUARE(5) 替换为 ((5)*(5))
#include <stdio.h>              // 包含标准头文件
#include "myheader.h"           // 包含自定义头文件
#ifdef DEBUG                    // 如果定义了 DEBUG 宏，则编译以下内容
    #ifndef WIN32               // 如果未定义 WIN32 宏，则编译以下内容
        #if VERSION > 2         // 如果 VERSION 大于 2，则编译以下内容
            // 这里可以放需要条件编译的代码
        #else                   // 如果条件不满足，则编译以下内容
            // 这里可以放条件不满足时的代码
        #endif                  // 结束 VERSION 条件编译
    #endif                      // 结束 WIN32 条件编译
#endif                          // 结束 DEBUG 条件编译
#undef MAX                      // 取消 MAX 宏定义
#pragma once                    // 防止头文件重复包含（仅限头文件使用）
#error "Unsupported platform"  // 生成编译错误
#warning "Deprecated function" // 生成编译警告
```

编译过程-编译

编译阶段简介

编译是 C/C++ 编译流程的第二步，由编译器（如 GCC、MSVC、Clang 等）负责。其核心任务是将经过预处理的源代码转化为汇编代码，为后续的汇编和链接阶段打下基础。编译不仅仅是”翻译”，还包含一系列严密的分析与优化流程，主要包括：

- **词法分析**：将源代码拆分为 Token（如关键字、标识符、常量、运算符等），并去除空白和注释。
- **语法分析**：根据 C/C++ 语法规则，将 Token 序列构建为语法树（AST），检查语法结构的正确性。
- **语义分析**：进一步检查代码含义，如变量是否声明、类型是否匹配、函数参数是否正确等。
- **中间代码生成与优化**：将语法树或 IR 转化为平台无关的中间代码（如三地址码、LLVM IR），并进行优化（如常量折叠、死代码消除、循环优化等），提高程序效率。
- **目标代码生成**：将优化后的中间代码翻译为特定平台的汇编代码，为汇编阶段做准备。
- **错误与警告报告**：在上述各阶段发现问题时，及时给出错误或警告，帮助开发者定位和修正代码。

编译过程-汇编

汇编阶段简介

汇编是 C/C++ 编译流程的第三步，由汇编器完成。其主要任务是将编译器生成的汇编代码转化为目标文件（如.obj 或.o 文件），为最终的链接阶段做准备。

汇编阶段的主要工作

- **目标文件生成**：将汇编代码转换为目标文件，包含机器指令和符号信息，供链接器使用。
- **错误与警告报告**：在汇编过程中发现语法或格式错误时，及时报告，便于开发者修正。

编译过程-链接

链接阶段简介

链接是 C/C++ 编译流程的最后一步，由链接器（Linker）负责。其核心任务是将多个目标文件（.o/.obj）与所需的库文件（静态库或动态库）合并，解决符号引用，生成最终的可执行文件或库文件。

链接阶段的主要工作

- **符号解析**：查找并匹配所有目标文件和库文件中的符号引用与符号定义，解决外部函数、全局变量等的引用关系。
- **重定位**：将各目标文件中的相对地址和引用调整为最终可执行文件中的绝对地址，确保程序运行时各部分能正确访问。
- **合并与优化**：合并相同的节（如代码段、数据段），去除未使用的符号，优化最终文件体积和加载效率。
- **错误与警告报告**：在符号未定义、重复定义、地址冲突等问题发生时，及时报告错误或警告，帮助开发者定位问题。

支持的编译器

Windows 平台

- **MinGW-w64**：GNU 编译器套件，免费开源
- **MSVC**：微软 Visual Studio 编译器，功能强大
- **Clang**：LLVM 编译器，编译速度快

跨平台

- **GCC**：Linux 系统标准编译器
- **Clang**：macOS 系统推荐编译器
- **Intel C++**：Intel 优化编译器，性能优秀

编译器选择建议

- **初学者**：MinGW-w64（简单易用，免费）
- **开发阶段**：MinGW-w64（快速编译，调试友好）
- **发布阶段**：MSVC（优化更好，性能更高）
- **跨平台**：GCC/Clang（兼容性好）

编译器配置详解

自动配置（推荐）

- **Qt 安装器自动配置**：安装时自动设置编译器路径
- **Qt Creator 自动检测**：启动时自动找到可用编译器

手动配置（高级）

- **工具 → 选项 → Kits**：配置编译套件
- **Qt 版本**：选择 Qt 库版本
- **编译器路径**：指定编译器可执行文件位置
- **调试器**：配置调试工具

配置验证

- **创建测试项目**：验证配置是否正确
- **编译测试**：确保能正常编译
- **运行测试**：确保能正常运行
- **调试测试**：确保调试功能正常

编译器配置示例

CMakeLists.txt 配置代码

```
# CMakeLists.txt 配置示例
cmake_minimum_required(VERSION 3.16)  # 指定所需的最低 CMake 版本
project(FirstApp)  # 定义项目名称
# 设置 C++ 标准为 17，并强制要求
set(CMAKE_CXX_STANDARD 17)                # 设置 C++ 标准为 C++17
set(CMAKE_CXX_STANDARD_REQUIRED ON)       # 强制使用指定的 C++ 标准
# 根据不同编译器设置编译选项
if(MSVC)
    # MSVC（微软编译器）下设置警告级别为 4
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /W4")
else()
    # GCC 或 Clang 下开启所有警告和额外警告
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")
endif()
# 查找并链接 Qt6 的 Core 和 Widgets 模块
find_package(Qt6 REQUIRED COMPONENTS Core Widgets)  # 查找 Qt6 核心和窗口部件模块
# 添加可执行文件，指定源文件
add_executable(FirstApp main.cpp)  # 生成名为 FirstApp 的可执行文件，源文件为 main.cpp
# 链接 Qt6 库到目标程序
target_link_libraries(FirstApp Qt6::Core Qt6::Widgets)  # 链接 Qt6 的 Core 和 Widgets 库
```

CMake 是什么?

CMake 是一个跨平台的自动化构建系统（build system）工具，主要用于管理 C/C++ 项目的编译过程。它通过编写 `CMakeLists.txt` 文件，描述项目的源文件、依赖、编译选项等，自动生成适合不同平台的本地构建文件（如 Makefile、Visual Studio 工程等）。

CMake 的主要特点

- **跨平台**：支持 Windows、Linux、macOS 等主流操作系统。
- **灵活性强**：可配置多种编译器和构建工具，适用于各种规模的项目。
- **模块丰富**：内置大量查找库和工具的模块，便于集成第三方库（如 Qt、Boost 等）。
- **自动化管理依赖**：通过 `find_package` 等机制自动查找和链接依赖库。
- **易于集成 IDE**：可为 Visual Studio、Qt Creator、CLion 等 IDE 生成工程文件。

CMake 基本工作流程

CMake 构建流程

1. **编写 CMakeLists.txt**：在项目根目录下创建 `CMakeLists.txt`，描述项目结构、依赖和编译规则。
2. **配置（Configure）**：运行 `cmake` 命令，生成本地构建系统文件（如 Makefile、.sln 等）。
3. **构建（Build）**：使用生成的构建文件（如 `make`、`ninja` 或 Visual Studio）进行编译和链接，得到可执行文件或库文件。

常用 CMake 命令

- `cmake -S . -B build` #配置项目，生成 build 目录
- `cmake --build build` #编译项目
- `cmake --install build` #安装（可选）

CMakeLists.txt 基本结构

CMakeLists.txt 常见内容

- `cmake_minimum_required(VERSION 3.16)` #指定 CMake 最低版本
- `project(MyProject)` #项目名称
- `set(CMAKE_CXX_STANDARD 17)` #指定 C++ 标准
- `find_package(Qt6 COMPONENTS Widgets REQUIRED)` #查找 Qt 库
- `add_executable(MyApp main.cpp)` #添加可执行文件
- `target_link_libraries(MyApp Qt6::Widgets)` #链接 Qt 库

建议

推荐将 CMakeLists.txt 文件和源代码、头文件、资源文件分目录管理，便于项目维护和扩展。

调试工具介绍

集成调试器

- **GDB**：常用的开源调试器，适用于 Linux 等系统。
- **LLDB**：LLVM 项目的调试器，常用于 macOS 和部分 Linux。
- **CDB**：微软命令行调试器，适用于 Windows 平台。
- **图形化调试界面**：如 Qt Creator 自带调试器，操作直观，适合初学者。

调试功能

- **断点设置和管理**：可以在代码的任意行设置断点，程序运行到断点时会暂停，便于分析程序状态。支持启用、禁用、删除断点等操作。
- **变量监视**：在调试过程中实时查看和修改变量的值，帮助定位变量赋值和变化过程中的问题。
- **调用栈查看**：显示函数调用层级关系，便于分析程序执行流程和定位函数调用链中的错误。
- **内存检查**：可以查看和修改内存中的数据，检测内存泄漏、越界访问等问题。
- **条件断点**：设置满足特定条件时才触发的断点，便于定位复杂逻辑下的错误。

Qt 调试示例

```cpp
#include <QApplication>  // Qt 应用程序主类
#include <QWidget>       // Qt 窗口部件基类
#include <QDebug>        // Qt 调试输出类

int main(int argc, char *argv[])
{
    // 创建 QApplication 对象，必须在任何 Qt 窗口部件之前创建
    // 在调试时，可以在此处查看 argc/argv 的值，了解程序启动参数
    QApplication app(argc, argv);
    // 创建一个 QWidget 窗口对象
    // 【调试建议】可在此行设置断点，调试时程序会在此暂停，便于单步跟踪后续窗口属性设置
    QWidget window;
    // 设置窗口标题
    // 在调试时，可以检查 window 对象的属性是否已正确设置
    window.setWindowTitle("调试示例");
    // 设置窗口大小
    window.resize(400, 300);
    // 输出调试信息到控制台
    // 【调试建议】可在调试器中观察此输出，确认窗口已成功创建
    qDebug() << "窗口创建成功";
    // 显示窗口
    window.show();
    // 进入 Qt 事件循环，等待用户操作
    // 【调试建议】可在此处监视 result 变量，查看应用程序的退出码
    int result = app.exec();
    // 输出应用程序退出时的返回值
    // 便于调试时确认程序是否正常退出
    qDebug() << "应用程序以返回值退出:" << result;
    // 返回应用程序的退出码
    return result;
}
```

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

什么是 Git?

版本控制的意义

- **版本控制**可以记录代码的每一次修改，方便回退、对比和协作。
- **Git** 是目前最流行的分布式版本控制系统，广泛应用于开源和企业项目。
- 使用 Git 可以有效防止代码丢失，便于团队协作开发。

核心术语

- **仓库（Repository）**：存放项目代码和历史记录的地方。
- **工作区（Working Directory）**：你当前编辑代码的本地文件夹。
- **暂存区（Stage/Index）**：临时保存即将提交的更改。
- **提交（Commit）**：将暂存区的内容保存到仓库历史中。
- **分支（Branch）**：代码开发的平行线，便于多人协作和功能开发。

Git 的下载安装

- 访问 Git 官网：`https://git-scm.com/downloads`
- 根据操作系统选择对应的安装包（Windows、macOS、Linux）。
- 以 Windows 为例，下载后双击安装包，按照安装向导逐步操作：
  - 选择安装路径（可使用默认设置）。
  - 选择组件（一般保持默认即可）。
  - 配置环境变量（建议选择” 将 Git 添加到 PATH”）。
  - 选择默认编辑器（可选 Notepad++、VS Code 等）。
  - 选择 HTTPS 传输方式（一般选择”Use the OpenSSL library”）。
  - 其他选项保持默认，点击”Install” 开始安装。
- 安装完成后，点击”Finish” 退出安装向导。

Git 的基本配置步骤

1. **首次使用 Git，需配置用户信息：**
   - `git config --global user.name Ÿour Name¨`
   - `git config --global user.email ÿour.email@example.com¨`
2. **配置默认编辑器**：`git config --global core.editor c̈ode --wait¨`
3. **查看配置**：`git config --list`

Git 的常用操作

- **初始化仓库**：`git init`
  在当前目录创建一个新的 Git 本地仓库。
- **克隆远程仓库**：`git clone 仓库地址`
  下载远程仓库到本地，并自动初始化。
- **查看当前状态**：`git status`
  显示工作区和暂存区的文件变动情况。
- **添加文件到暂存区**：`git add 文件名`
  将指定文件的更改加入暂存区，准备提交。
- **批量添加所有更改**：`git add .`
  一次性将所有变动文件加入暂存区。
- **提交更改**：`git commit -m " 提交说明"`
  将暂存区内容保存到本地仓库历史。
- **查看提交历史**：`git log`
  按时间顺序显示提交记录。
- **创建新分支**：`git branch 分支名`
  新建一个分支，便于并行开发。
- **切换分支**：`git checkout 分支名`
  切换到指定分支。
- **合并分支**：`git merge 分支名`
  将指定分支的更改合并到当前分支。
- **推送到远程仓库**：`git push`
  将本地分支的提交推送到远程仓库。
- **拉取远程更新**：`git pull`
  获取远程仓库最新内容并合并到本地。
- **解决冲突**：手动编辑冲突文件，保存后用 `git add 文件名`标记为已解决，再继续提交。

目录

1 编程概念入门
2 软件开发概述
3 Qt 开发环境安装
4 Qt Creator IDE
5 编译器和调试配置
6 版本控制集成
7 项目实践
8 其他 IDE 配置

常见 Qt 项目模板

应用程序模板

- **Qt Widgets Application**：传统桌面窗口程序，适合大多数入门和实际项目。
- **Qt Quick Application**：基于 QML 的现代界面应用，适合动画和移动端开发。
- **Console Application**：命令行程序，适合学习 C++ 基础和算法。
- **Qt for Python Application**：使用 Python 语言开发 Qt 应用，语法简单，易于入门。

库模板

- **C++ Library**：通用 C++ 静态或动态库。
- **Qt Widgets Library**：基于 Qt Widgets 的自定义控件库。
- **Qt Quick Library**：QML 模块或自定义 QML 控件库。
- **Plugin Library**：插件开发模板，便于扩展 Qt 应用功能。

第一个 Qt Widgets 项目

项目结构说明

- **main.cpp**：程序入口，负责应用初始化和主窗口显示。
- **CMakeLists.txt / .pro 文件**：项目构建配置文件。
- **其他源文件**：可根据需要添加窗口类、资源文件等。

代码文件已放在 `code/`文件夹中，便于查阅和复用。

主要步骤

1. 新建”Qt Widgets Application” 项目，命名如 `HelloQt`。
2. 选择合适的构建工具（如 CMake 或 qmake）。
3. 编写或替换 `main.cpp`，实现主窗口和控件。
4. 构建并运行，看到窗口和控件效果。

环境验证步骤

1. 编译 FirstApp 程序
2. 运行程序验证
3. 测试调试功能
4. 验证版本控制
5. Qt Creator 和 VS Code 的配置

常见问题解决

- 编译器路径配置
- Qt 库路径设置
- 环境变量配置
- 权限问题处理

目录

1. 编程概念入门
2. 软件开发概述
3. Qt 开发环境安装
4. Qt Creator IDE
5. 编译器和调试配置
6. 版本控制集成
7. 项目实践
8. 其他 IDE 配置

Visual Studio 简介

Visual Studio 概述

Visual Studio 是微软开发的集成开发环境，是 Windows 平台上最强大的 IDE 之一，支持多种编程语言和框架，包括 C++ 和 Qt 开发。

版本选择

- **Community 版**：免费，适合个人和小团队
- **Professional 版**：付费，适合专业开发
- **Enterprise 版**：付费，适合大型企业

安装要求

- **操作系统**：Windows 10/11
- **内存**：至少 8GB RAM（推荐 16GB）
- **硬盘空间**：至少 20GB 可用空间
- **网络连接**：下载和安装需要稳定网络

Visual Studio 安装配置

1. **下载 Visual Studio Installer**
   - 访问 visualstudio.microsoft.com
   - 下载 Visual Studio Installer
   - 选择 Community 版本（免费）
2. **选择工作负载**
   - 选择”使用 C++ 的桌面开发”
   - 选择”使用 C++ 的游戏开发”（可选）
   - 确保包含 MSVC 编译器
3. **安装 Qt VS Tools**
   - 在扩展管理器中搜索”Qt Visual Studio Tools”
   - 下载并安装该扩展
   - 重启 Visual Studio
4. **配置 Qt 路径**
   - 在 Qt VS Tools 中配置 Qt 安装路径
   - 选择 Qt 版本和编译器
   - 验证配置是否正确

推荐配置

- **MSVC 编译器**：Visual Studio 自带
- **Qt 6.x 版本**：最新稳定版本
- **CMake 支持**：现代 C++ 项目构建
- **调试工具**：集成调试器

安装时间

- **下载时间**：1-3 小时
- **安装时间**：30 分钟-2 小时
- **配置时间**：15-30 分钟

Visual Studio 中 Qt 开发流程

Visual Studio 中 Qt 项目开发流程

- **使用 Qt VS Tools**：通过扩展创建 Qt 项目，支持 Qt Widgets、Qt Quick 等多种项目类型
- **配置项目**：选择 Qt 版本和编译器，设置包含路径、库路径等项目属性
- **编写代码与界面设计**：在 Visual Studio 中编写 C++ 代码，使用 Qt Designer 或代码方式设计 UI
- **编译与调试**：利用 Visual Studio 强大的编译和调试工具进行开发
- **部署与发布**：生成可执行文件和安装包，完成项目发布

优势特点

- **调试体验**：集成 Visual Studio 强大的断点、单步、变量监视等调试功能
- **性能分析**：支持性能分析工具，便于优化 Qt 应用

VS Code 简介

VS Code 概述

Visual Studio Code 是微软开发的轻量级代码编辑器，具有 IDE 的强大功能，但保持了编辑器的轻量和快速特性，非常适合现代软件开发。

VS Code 优势

- **轻量级**：启动快速，占用资源少
- **免费开源**：完全免费，源代码开放
- **扩展丰富**：大量插件和扩展
- **跨平台**：Windows、macOS、Linux
- **现代化**：界面美观，用户体验佳
- **智能提示**：智能提示，提高开发效率

适用场景

- **小型项目**：快速开发和原型设计
- **多语言开发**：支持几乎所有编程语言
- **学习编程**：轻量级，适合初学者
- **智能提示**：智能提示，提高开发效率

VS Code 安装和基础配置

1. **下载安装 VS Code**
   - 下载适合操作系统的版本
   - 安装并启动 VS Code
2. **安装必要扩展**
   - C/C++ Extension Pack
   - CMake Tools
   - Qt tools
   - GitLens（可选）
3. **配置 C++ 环境**
   - 安装 MinGW 或 MSVC 编译器
   - 配置编译器路径
   - 设置 IntelliSense
4. **配置 Qt 环境**
   - 设置 Qt 安装路径
   - 配置 CMake 查找 Qt
   - 验证配置

推荐扩展

- **C/C++**：Microsoft 官方扩展
- **CMake Tools**：CMake 项目支持
- **Qt tools**：Qt 开发支持
- **GitLens**：Git 集成增强
- **Code Runner**：快速运行代码

VS Code 中 Qt 项目结构与管理

项目结构建议

- **使用 CMake**：推荐采用 CMake 作为 Qt 项目的构建系统，便于跨平台和自动化管理。
- **文件夹组织**：建议将源代码（`src`）、头文件（`include`）、资源文件（`resources`）、构建目录（`build`）等分门别类，结构清晰。
- **版本控制**：集成 Git，使用 `.gitignore` 忽略构建产物和 IDE 配置文件，便于多人协作和代码管理。
- **多文件编辑**：VS Code 支持分屏、标签页，便于同时编辑多个文件，提高开发效率。
- **依赖管理**：通过 CMake 的 `find_package` 等机制管理 Qt 及第三方库依赖。

VS Code 中 Qt 开发详细流程

开发流程详解

1. **创建项目**：新建 CMake 工程，配置 Qt 相关参数，初始化 Git 仓库。
2. **编写代码**：利用 VS Code 的智能补全、语法高亮、错误提示等功能高效编写 C++/Qt 代码。
3. **构建项目**：使用 CMake Tools 插件一键配置和编译，支持 Debug/Release 等多种构建类型。
4. **调试程序**：集成 GDB/LLDB/MSVC 等调试器，支持断点、单步、变量监视等调试操作。
5. **运行测试**：可集成 Google Test、Catch2 等单元测试框架，自动化测试代码质量。
6. **版本提交**：通过 Git 插件进行代码提交、分支管理、冲突解决等操作。

总结

本章要点

- 了解 Qt 开发环境的组成
- 掌握 Qt Creator 的安装和配置
- 学会配置编译器和调试工具
- 熟悉版本控制集成
- 能够创建和运行 Qt 项目
- 了解版本兼容性和学习路径

课后任务

- 安装 Qt Creator
- 安装 VS Code
- 配置 Qt Creator 和 VS Code
- 完成 FirstApp 程序的编译和运行
- 熟悉 Qt 开发环境
- 自学 CMake
- 自学 Git