第 3 章

STM32CubeIDE 集成开发环境

本章重点讲述 STM32CubeIDE 集成开发环境的使用方法和具体操作细节。首先，介绍 STM32CubeIDE 的基本概念及其安装准备工作，包括软件包获取、安装流程、启动软件和打开项目，确保读者掌握基础操作。

在界面功能和操作部分，本章详细解析 C/C++开发环境中的主要视图、工具栏功能以及文本编辑器的操作，使读者能够高效利用 STM32CubeIDE 进行代码编写和调试；同时，介绍 STM32CubeMX 生成项目的文件组成，解释包括 CMSIS 驱动程序文件、HAL 驱动程序文件、用户程序文件、启动文件及根目录下的文件的具体内容，并讲解 Include 搜索路径的设置。

本章通过说明 File、Edit、Source、Refactor、Navigate、Search、Project、Run、Window 和 Help 菜单的各项功能，使读者全面掌握 STM32CubeIDE 的多样化操作方法。

在实际操作部分，涵盖了新建、导入、管理、切换、删除和导出工程，以及固件库管理、代码编译、调试和运行配置、启动调试等，逐步指导读者完成开发全过程。

此外，本章还指导读者如何使用内置的 STM32CubeMX 进行项目创建、MCU 配置和代码生成，提供图形化配置的便利性。最后，介绍 STM32CubeIDE 的使用偏好设置以及推荐的 STM32F407 开发板和仿真器选择，为读者实际开发提供硬件参考和优化方案。

通过本章的学习，读者将全面了解 STM32CubeIDE 的安装与使用，从而大幅提升基于 STM32 微控制器的开发效率。

3.1 STM32CubeIDE 基本介绍

STM32CubeIDE 是 STM32Cube 生态系统中的一个重要软件工具，是 ST 官方免费提供的 STM32 MCU/MPU 程序开发 IDE 软件。ST 公司最初并没有自己的 STM32 开发 IDE 软件，为了完善 STM32Cube 生态系统中这重要的一环，ST 公司在 2017 年年底收购了 Atollic 公司，将专业版 TrueSTUDIO 改为免费的。2019 年 4 月，ST 公司正式推出了 STM32CubeIDE 1.0.0。

STM32CubeIDE 就是在 TrueSTUDIO 的基础上改进和升级得来的，具有如下特点。

（1）STM32CubeIDE 使用 Eclipse IDE，具有强大的编辑功能，其使用习惯与 TrueSTUDIO 相同。

（2）STM32CubeIDE 使用 GNU C/C++编译器，支持在 STM32 项目开发中使用 C++编程。

（3）STM32CubeIDE 内部集成了 STM32CubeMX，在 STM32CubeIDE 里就可以进行 MCU 图形化配置和代码生成，然后在初始代码的基础上继续编程。当然，STM32CubeIDE 也可以和独立的 STM32CubeMX 配合使用。

正式推出 STM32CubeIDE 后，ST 公司就不再更新 TrueSTUDIO 了，新的设计推荐使用 STM32CubeIDE。

用户可以从 ST 公司网站下载最新版 STM32CubeIDE 的安装文件。安装文件中只有一个可执行文件，双击运行就可以开始安装。

为提升功能丰富且高能效的 STM32 系列微控制器的易用性，2019 年，ST 公司在 STM32Cube 软件生态系统中增加了一个免费的多功能 STM32 开发工具——STM32CubeIDE。

STM32CubeIDE 是 ST 官方提供的免费软件开发工具，也是 STM32Cube 生态系统的核心。它基于 Eclipse/CDT 框架、GCC 编译工具链和 GDB 调试工具，支持添加第三方功能插件。同时，STM32CubeIDE 还集成了部分 STM32CubeMX 和 STM32CubeProgrammer 的功能，是一个“多合一”的 STM32 开发工具。STM32CubeIDE 架构如图 3-1 所示。

图 3-1　STM32CubeIDE 架构

用户只需要 STM32CubeIDE 这一个工具，就可以完成从芯片选型、项目配置、代码生成，到代码编辑、编译、调试和烧录的所有工作。

STM32CubeIDE 基于 Eclipse 的框架，它继承了 Eclipse 所特有的特性，如工作空间、透视图等。STM32CubeIDE 软件界面如图 3-2 所示。

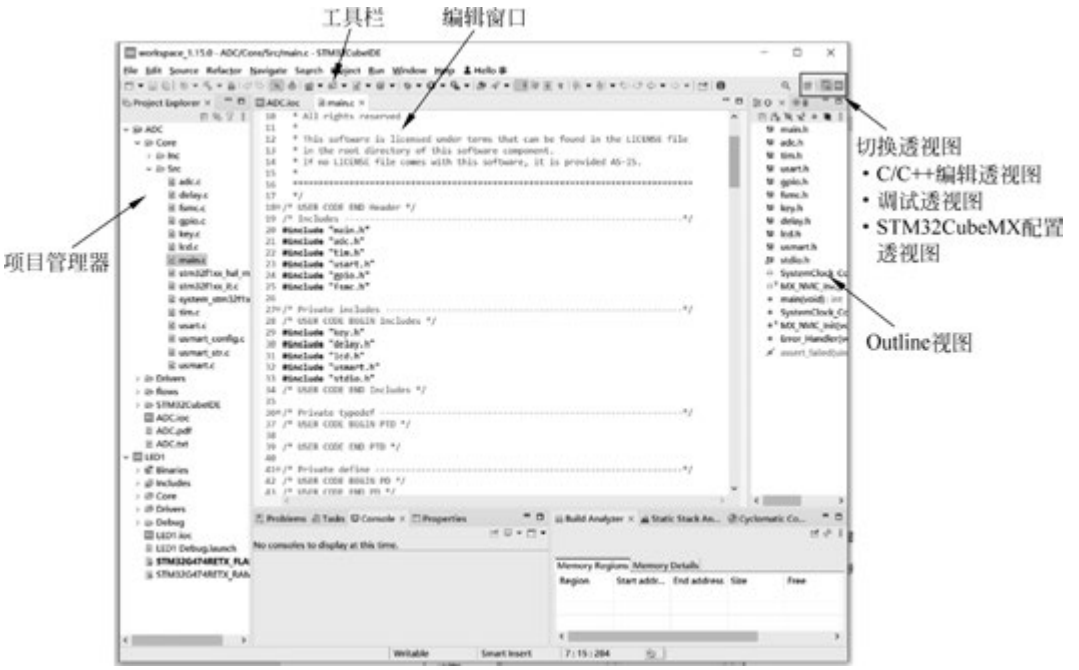

图 3-2 STM32CubeIDE 软件界面

第 9 集
微课视频

3.2 STM32CubeIDE 使用前的准备

在使用 STM32CubeIDE 之前，开发者需要进行一系列准备工作以确保开发环境的顺利搭建和项目的成功实施。详细步骤如下。

1）硬件准备

（1）开发板。选择适合项目需求的 STM32 开发板，如 Nucleo、Discovery 或 Evaluation 板。

（2）调试工具。确保有合适的调试器，如 ST-LINK/V2。

（3）连接线。USB 线用于连接开发板和计算机。

2）软件准备

（1）安装 STM32CubeIDE。从 STMicroelectronics 官方网站下载最新版本的 STM32CubeIDE 安装包，并根据操作系统（Windows、macOS 或 Linux）的指引进行安装。

（2）下载并安装驱动程序。安装 ST-LINK USB 驱动程序，以确保调试器和计算机之间的通信正常。

3）必要配置

（1）安装 STM32CubeMX。虽然 STM32CubeIDE 内置了 STM32CubeMX 功能，但确保 STM32CubeMX 插件是最新版本，有助于生成初始化代码和配置外设。

（2）获取相关固件包。在 STM32CubeIDE 中，通过 Help→Manage embedded software

packages 菜单命令下载和管理对应的 STM32 固件包。

4）开发板设置

（1）接电与连接。将开发板通过 USB 线连接到计算机，并确认跳线设置正确。

（2）调试器配置。打开 STM32CubeIDE 并确保能识别到连接的调试器 ST-LINK/V2。

5）初步项目配置

（1）启动 STM32CubeIDE。打开 STM32CubeIDE 并创建一个新工作空间以存放项目文件。

（2）新建项目。使用 STM32CubeMX 创建一个新项目，选择对应的 STM32 芯片或开发板，配置时钟和外设，并生成初始化代码。

（3）导入 IDE。STM32CubeMX 生成的项目自动导入 STM32CubeIDE。

6）编译与调试准备

（1）编译器配置。确保 GNU Arm Embedded Toolchain 正确安装并配置在 STM32CubeIDE 中。

（2）项目编译。在 STM32CubeIDE 中完成项目编译，确认无编译错误。

（3）调试设置。配置调试器选项，确保能够进行单步调试和断点设置。

7）学习与实践

熟悉 STM32CubeIDE 的用户手册和参考文档，了解各项功能和使用技巧。

8）示例项目

参考 STM32Cube 提供的示例项目，帮助快速上手和理解 STM32 开发流程。

通过以上步骤，开发者可以为使用 STM32CubeIDE 做好充分准备，建立一个高效、可靠的开发环境。

3.2.1 STM32CubeIDE 软件包获取

在安装 STM32CubeIDE 之前，首先登录 ST 公司官网（https://www.st.com.cn/zh/development-tools/stm32cubeide.html#），选择 STM32CubeIDE 安装包版本（如 STM32CubeIDE1.15.1）。如图 3-3 所示，单击“获取软件”按钮，进入 STM32CubeIDE 安装包下载许可协议选择页面，需要登录 MyST，与 STM32CubeMX 的操作方式相同，这里从略。

下载到的 STM32CubeIDE 安装包如图 3-4 所示。

3.2.2 STM32FCubeIDE 的安装

STM32FCubeIDE 安装步骤如下。

将 STM32CubeIDE 安装包解压缩后，得到如图 3-5 所示的 STM32CubeIDE 应用程序。

双击 st-stm32cubeide_1.15.1_21094_20240412_1041_x86_64.exe 应用程序，弹出如图 3-6 所示的 STM32CubeIDE 安装向导。

图 3-3 STM32CubeIDE 安装包下载页面

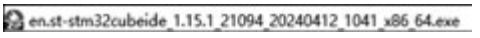

图 3-4 STM32CubeIDE 安装包

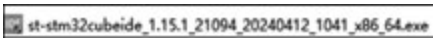

图 3-5 STM32CubeIDE 应用程序

图 3-6 STM32CubeIDE 安装向导

单击 Next 按钮，弹出如图 3-7 所示的 License Agreement(许可协议)界面。

单击 I Agree 按钮，弹出如图 3-8 所示的 Choose Install Location 界面，选择安装路径。

图 3-7 License Agreement(许可协议)界面

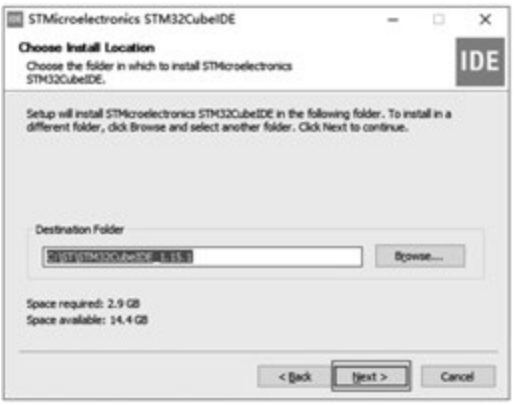

图 3-8 Choose Install Location 界面

一般选择默认路径，单击 Next 按钮，弹出如图 3-9 所示的 Choose Components(选择组件)界面。

单击 Install 按钮，弹出如图 3-10 所示的 Installing(安装)界面。

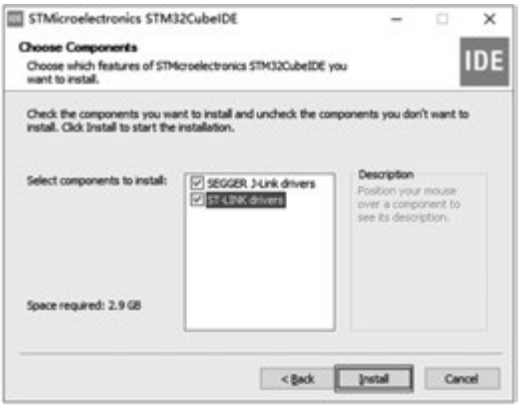

图 3-9 Choose Components(选择组件)界面

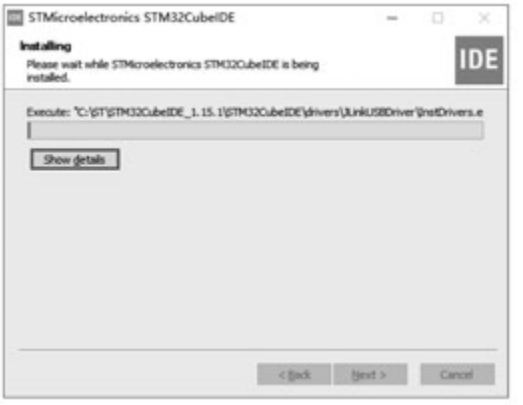

图 3-10 Installing(安装)界面

在 STM32CubeIDE 安装过程中，会弹出如图 3-11 所示的 STMicroelectronics 通用串行总线设备安装选择对话框。

由于在应用 STM32CubeIDE 开发调试时，要将 ST-LINK/V2 插入计算机的 USB 接口，所以单击“安装”按钮，返回 Installing 界面继续安装，如图 3-12 所示。

STM32CubeIDE 安装完成，弹出如图 3-13 所示的 Installation Complete 界面。

单击 Next 按钮，弹出如图 3-14 所示的创建桌面快捷方式界面。

勾选 Create desktop shortcut 复选框，单击 Finish 按钮，在计算机桌面生成如图 3-15 所示的 STM32CubeIDE 快捷方式。

图 3-11 STMicroelectronics 通用串行总线设备安装选择对话框

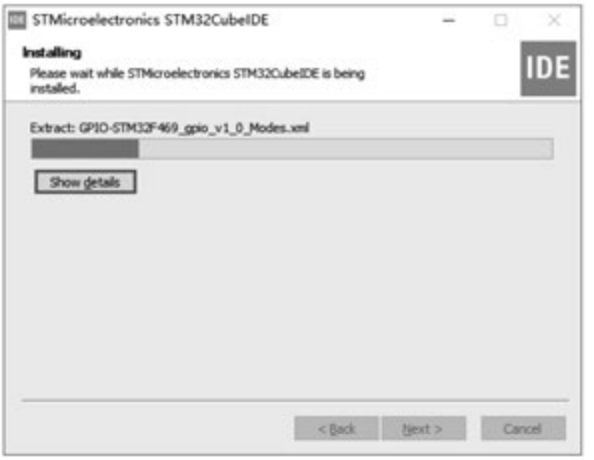

图 3-12 继续安装

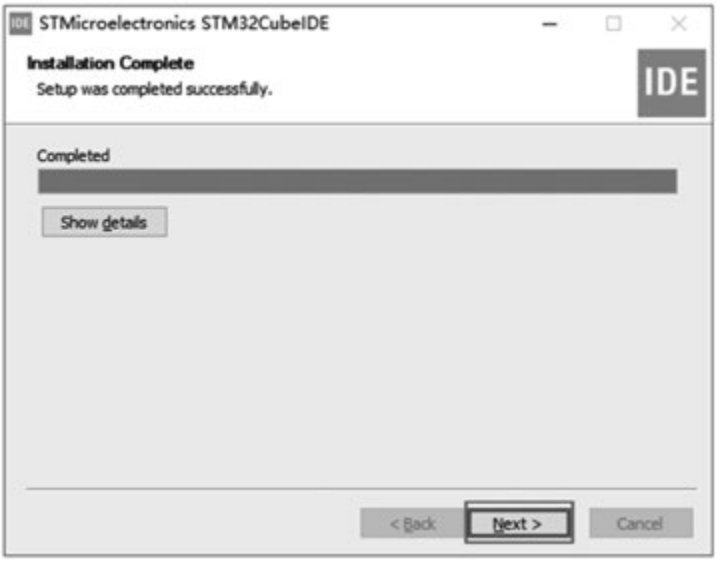

图 3-13 Installation Complete 界面

图 3-14　创建桌面快捷方式界面

图 3-15　STM32CubeIDE 快捷方式

3.2.3　启动软件

双击 STM32CubeIDE 快捷方式启动软件。

在 STM32CubeIDE 启动时，会弹出如图 3-16 所示的 STM32CubeIDE Launcher 对话框，要求设置一个工作空间目录。

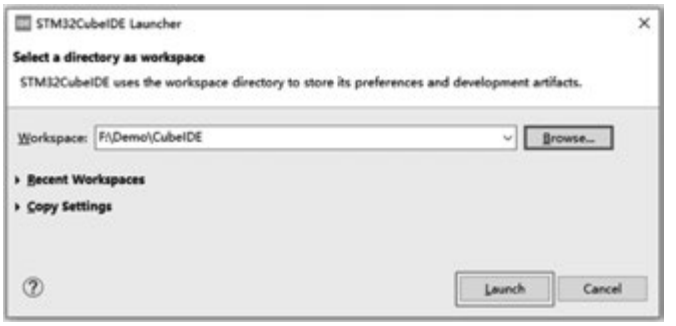

图 3-16　STM32CubeIDE Launcher 对话框

STM32CubeIDE 的工作空间(Workspace)是一个包含项目、设置和资源的文件系统目录。工作空间的主要功能和特点如下。

(1) 项目管理。工作空间是所有项目存储的地方，可以包含多个不同的项目。每个项目都包含特定的源代码、配置文件和其他相关的资源。

(2) 独立配置。每个工作空间都有其独立的配置文件，这些文件保存了与具体开发环境相关的设置，如构建配置、调试配置、编辑器设置等。

(3) 资源组织。工作空间提供了一种组织和管理项目资源的方式，帮助开发者方便地浏览和查找项目文件夹和文件。

(4) 插件和扩展。与工作空间关联的配置文件中还包含了各种插件和扩展的设置和数据，这些插件扩展了 IDE 的功能。

(5) 灵活性。开发者可以创建并切换多个工作空间,以适应不同的开发需求或项目。这允许他们在不同项目或任务间彼此独立地管理和跟踪设置。

简而言之,STM32CubeIDE 的工作空间是一个为开发者提供组织、管理和定制开发环境的基础架构,使他们能够高效地处理多个项目和资源。

例如,一些示例项目都放在 F:\Demo\CubeIDE 目录下,这个目录就是工作空间目录。启动 STM32CubeIDE 后,设置工作空间(Workspace)目录,如图 3-16 所示。单击 Browse 按钮选择这个目录,然后单击 Launch 按钮启动软件。

另外,也可以执行 File→Switch Workspace 菜单命令切换工作空间,如图 3-17 所示。

图 3-17 切换工作空间

启动 STM32CubeIDE 打开一个新的工作空间后,会显示如图 3-18 所示的信息中心页面。这个页面中有创建 STM32CubeIDE 项目的快捷按钮,这些功能的具体操作在后面会介绍。

图 3-18 信息中心页面

(1) Start new STM32 project:开始创建一个新的 STM32 项目。

(2) Start new project from STM32CubeMX file(.ioc):从 STM32CubeMX 的 *.ioc 文件开始创建一个项目。

(3) Import project：导入 STM32 工程项目。

(4) Import STM32Cube example：导入 STM32Cube 示例。

信息中心页面右侧的 Support & Community 区域有一些支持和社区网站的链接，单击后可在系统默认的浏览器中打开。

Quick links 区域有一些技术资料的 PDF 文档或 HTML 网页的链接，单击 Read STM32CubeIDE Documentation 后会打开一个文档列表页面，有更多有用的技术文档，包括 STM32CubeIDE 的用户手册等，用户在编程时可以查阅这些资料文档。

3.2.4 打开项目

本节以项目 ADC 为例，讲解 STM32CubeIDE 软件的基本使用方法以及 STM32CubeIDE 项目的文件组成。

首先关闭信息中心页面，然后执行 File→Open projects from file system 菜单命令，会弹出如图 3-19 所示的 Import Projects from File System or Archive 对话框，这个对话框用于将一个项目导入当前工作空间中。

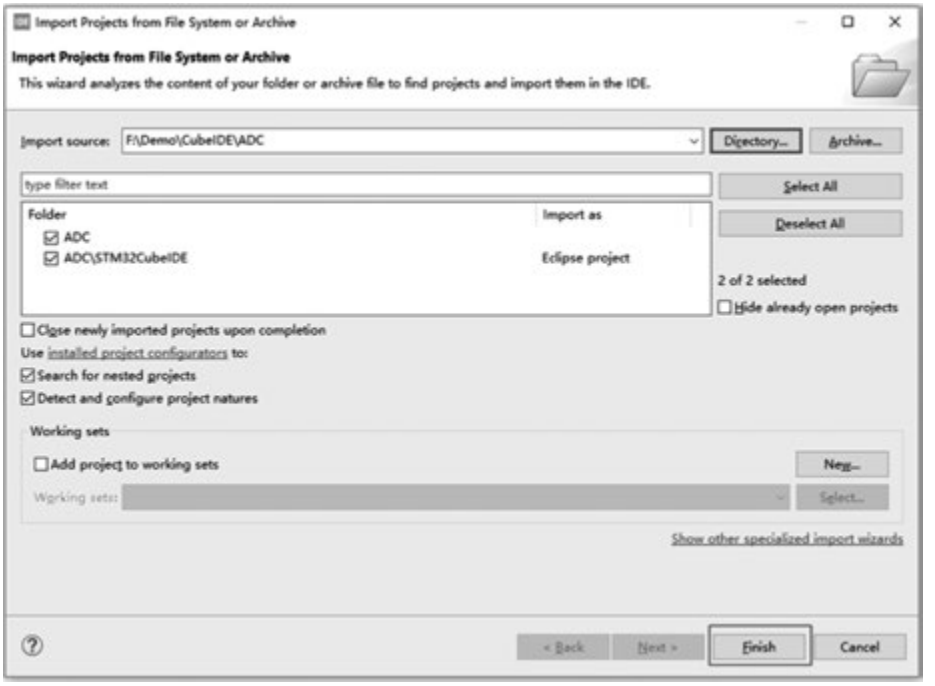

图 3-19 Import Projects from File System or Archive 对话框

在 Import Projects from File System or Archive 对话框中，首先单击 Directory 按钮，选择示例目录下项目 ADC 的根目录。选择后会在 Import source 文本框里显示此目录，并将项目名称显示在下方的列表里。其他设置保持默认设置，最后单击 Finish 按钮，就可以打开项目 ADC 了。

打开项目 ADC 后的 STM32CubeIDE 界面如图 3-20 所示。

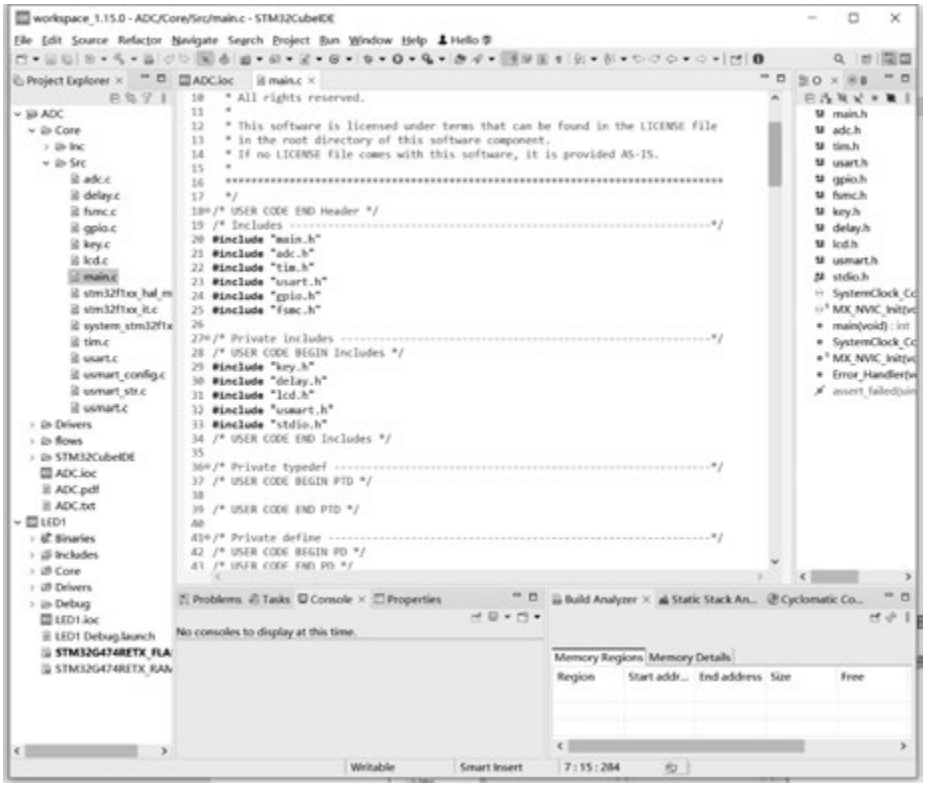

图 3-20 打开项目 ADC 后的 STM32CubeIDE 界面

3.2.5 STM32CubeIDE 的一些基本概念

STM32CubeIDE 是基于 Eclipse 的 IDE 软件，与一般的编程开发 IDE 界面类似。如果读者对 Eclipse 比较熟悉，那么对 STM32CubeIDE 的操作也就很容易上手了；如果没有使用过 Eclipse，需要对 Eclipse 中的一些基本概念有所了解。

Eclipse 是一个使用广泛地编程开发 IDE，通过使用插件可以支持不同编程语言的开发，如 C/C++、Java、Python 等。Eclipse 的项目管理和界面组织有一些基本的概念，包括工作空间、项目、视图、场景。

1. 工作空间

STM32CubeIDE 通过工作空间（Workspace）对工程进行管理，打开 STM32CubeIDE 时，会新建一个默认的工作空间，用户也可以通过单击 Browse 按钮另外选择一个文件夹作为工作空间，之后新建或导入的工程就都属于前面选择的这个工作空间。同一个工作空间下的工程具有相同的 IDE 层面的配置（通过 Window→Preferences 菜单命令进行设置），如显示和编辑的风格设置等。

工作空间是多个项目(Project)的集合,一个工作空间对应一个实际的目录。例如,在图 3-16 中设置了一个工作空间为目录 F:\Demo\CubeIDE,这个目录下的子目录的构成如图 3-21 所示。

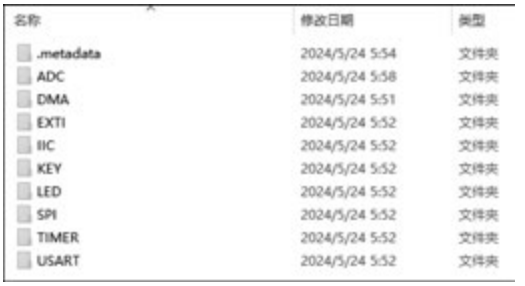

图 3-21 一个工作空间的目录组成

在这个工作空间目录下,有 STM32CubeIDE 为工作空间自动创建的. metadata 文件夹,用于存放工作空间 IDE 的各种配置文件,以及工作空间所管理的项目的信息文件。

在工作空间目录下,可以有多个 STM32CubeIDE 项目文件夹,STM32CubeIDE 项目名称就是文件夹名称。

一个工作空间管理的项目最好就放在这个工作空间的目录下,虽然也可以将非工作空间目录下的项目导入工作空间中。

在 STM32CubeIDE 中,任何时候只能打开一个工作空间。用户可以在 STM32CubeIDE 启动时的对话框里选择工作空间,也可以在运行时通过执行 File→Switch Workspace 菜单命令切换当前工作空间。

2. 项目(Project)

一个 STM32CubeIDE 项目(Project)就是一个文件夹下的所有子目录和文件的集合,项目的名称就是文件夹的名称。一个项目包含很多文件和子目录。例如,项目 1-LED 根目录下的文件和子目录构成如图 3-22 所示,项目 1-LED/STM32CubeIDE 根目录下的文件和子目录构成如图 3-23 所示。项目目录下的子目录/. settings 是自动生成的用于管理项目信息的子目录,几个没有名称只有扩展名的文件是项目管理的相关文件,如. cproject、. mxproject 和. project。LED. ioc 是 STM32CubeMX 项目文件。其他文件和子目录就是 STM32 编程相关的用户程序文件和驱动程序文件。

图 3-22 项目 1-LED 根目录下的文件和子目录构成

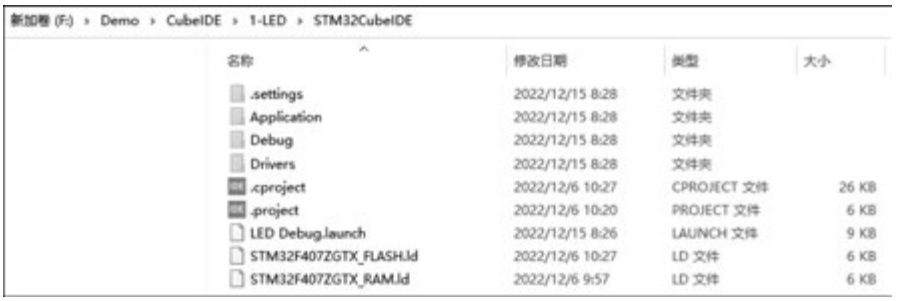

图 3-23 项目 1-LED/STM32CubeIDE 根目录下的文件和子目录构成

1）.project 文件和.cproject 文件

在 STM32CubeIDE 中，.project 文件和.cproject 文件是 Eclipse 工程的一部分，因为 STM32CubeIDE 基于 Eclipse 平台。这些文件包含了项目的不同配置和设置。

.project 文件主要描述工程的通用信息和元数据，包括工程名称、工程类型、与工程相关的配置信息插件、构建配置等信息。在 Eclipse 中打开工程时，会读取这个文件识别工程以及加载必要的插件和工具。

.cproject 文件主要描述与 C/C++开发相关的配置信息，包括编译器、链接器的设置，以及源码路径，包含路径、宏定义、构建变量等详细信息。在编译和链接工程时，Eclipse 会参考这个文件中的设置以确定如何处理源代码和生成最终的可执行文件。

也就是说，.project 文件是描述工程元数据和通用设置的文件；.cproject 文件是描述与 C/C++开发相关详细配置的文件。这两个文件相辅相成，确保 STM32CubeIDE 能够正确地理解和构建你的工程。

2）.launch 文件

图 3-23 中，有一个 LED Debug.launch 文件。LED Debug.launch 文件通常是一个基于 Eclipse IDE 的启动配置文件，它专门为调试一个 LED 控制项目而创建。此类文件包含了与调试该项目相关的详细配置，如目标板信息、调试器设置、硬件接口等。

在 Eclipse 平台上（包括基于 Eclipse 的 IDE，如 STM32CubeIDE、RT-Thread Studio 等），.launch 文件主要用于保存特定项目的运行配置。这些文件通常包含运行或调试应用程序所需要的信息。

.launch 文件的作用如下。

（1）保存运行配置。.launch 文件保存了如何启动、运行或调试项目的详细配置，包括启动程序的入口点、必要的环境变量、程序参数、工作目录等信息。

（2）支持多种启动模式。可以创建多个.launch 文件以支持不同的启动模式。例如，一个.launch 文件可以用来在本地调试程序，另一个.launch 文件可以用来在远程服务器上运行同一个程序。

（3）简化项目运行。因为.launch 文件记录了所有需要的配置信息，可以通过单击按钮快速启动应用程序，而无须每次手动输入参数或配置环境。

.launch 文件包含的信息如下。

(1) 项目名称：与该配置相关的 Eclipse 项目名称。

(2) 主类或启动文件：指定要运行的入口类(对于 Java 项目)或启动文件。

(3) VM 参数或程序参数：用于指定虚拟机或应用程序所需的参数。

(4) 环境变量：运行该项目所需的环境变量。

(5) 工作目录：项目运行时所使用的工作目录。

(6) 调试配置：如果用于调试，文件还会包含调试相关的设置，如断点信息、日志文件等。

.launch 文件用于保存和管理项目的运行和调试配置，是 Eclipse 平台及其衍生 IDE 中的一个重要功能。这些文件帮助开发者快速、方便地在各种配置下启动和调试项目，提升了开发效率。

3. 视图

STM32CubeIDE 界面上有很多子窗口，这些子窗口称为视图(View)。例如，界面左侧显示项目目录和文件组成的 Project Explorer 视图，界面右侧显示文件概览的 Outline 视图。一个视图就是实现一些功能的窗口，通常显示在一个多页组件上，右上角有关闭视图的按钮。

STM32CubeIDE 是功能强大的 IDE，有很多视图，用户可根据需要选择显示各种视图。执行 Window→Show View 菜单命令，会显示如图 3-24 所示的子菜单，其中会显示一些常用的视图。例如，SFRs 是显示 MCU 的特殊寄存器内容的视图，在调试程序时有用；Outline 是显示一个文件内代码概览的视图，可以显示文件内的函数、宏、类型、变量等各种定义，在浏览程序时特别有用。

如果单击最后的 Other 菜单项，会弹出如图 3-25 所示的 Show View 对话框，这个对话框里分类列出了所有视图。

图 3-24 常用视图菜单

图 3-25 Show View 对话框

4. 场景

STM32CubeIDE 的视图非常多，都显示出来会很杂乱，且在工作状态切换时逐个打开或关闭视图效率低下。例如，编程状态和调试状态要用到不同的视图。因此，Eclipse 使用场景(Perspective)管理视图。场景就是多个视图组成的一种工作界面，一个场景一般对应于一种工作需求，例如，C/C++场景是最常用的场景，图 3-20 显示的就是这个场景；Debug 场景是用于程序调试时的工作场景。

执行 Window→Perspective 菜单命令，会显示如图 3-26 所示的子菜单。单击 Customize Perspective 菜单项，可以打开一个对话框对当前场景进行定制，定制内容包括工具栏按钮和菜单项的可见性，可以保存定制的场景并自定义场景名称。

在图 3-26 中，单击 Perspective→Open Perspective→Other 菜单项，会弹出如图 3-27 所示的 Open Perspective 对话框，其中有 STM32CubeIDE 预定义的场景。在工作状态变化时，场景一般会自动切换。例如，STM32CubeIDE 启动后就处于 C/C++场景，这是最常用的场景；如果在项目管理器里双击 STM32CubeMX 文件 ADC. ioc，会自动切换到 Device Configuration Tool 场景，也就是内置的 STM32CubeMX 操作界面；如果下载程序开始调试，会自动切换到 Debug 场景。

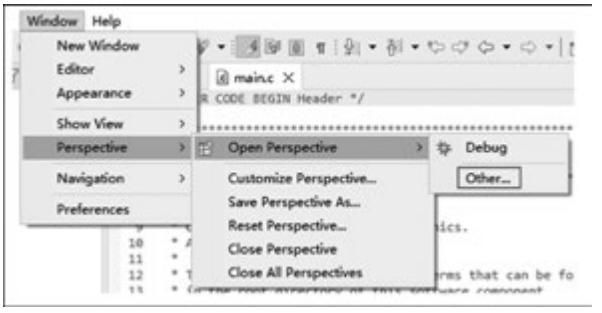

图 3-26 场景管理子菜单

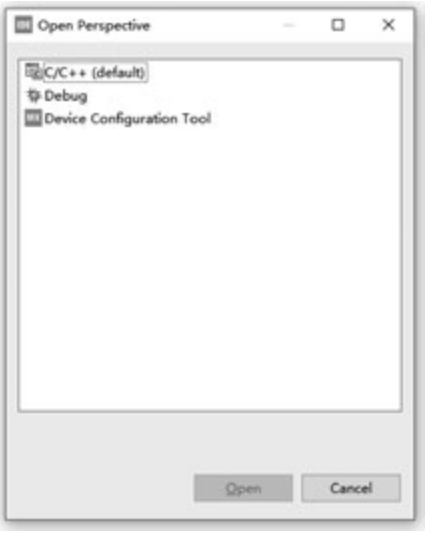

图 3-27 Open Perspective 对话框

3.2.6 STM32Cube 软件库设置

STM32CubeIDE 集成了 STM32CubeMX 的功能，因为需要用到 STM32Cube MCU 固件库和扩展库(统称为软件库)，所以要设置软件库的存储路径和升级方式。执行 Window→Preferences 菜单命令，弹出 Preferences 对话框。用户在这个对话框里可以对软件的各种特性进行设置，包括界面配色方案、字体等。STM32Cube 软件库相关的设置在 STM32Cube→

Firmware Updater 页面，设置结果如图 3-28 所示。

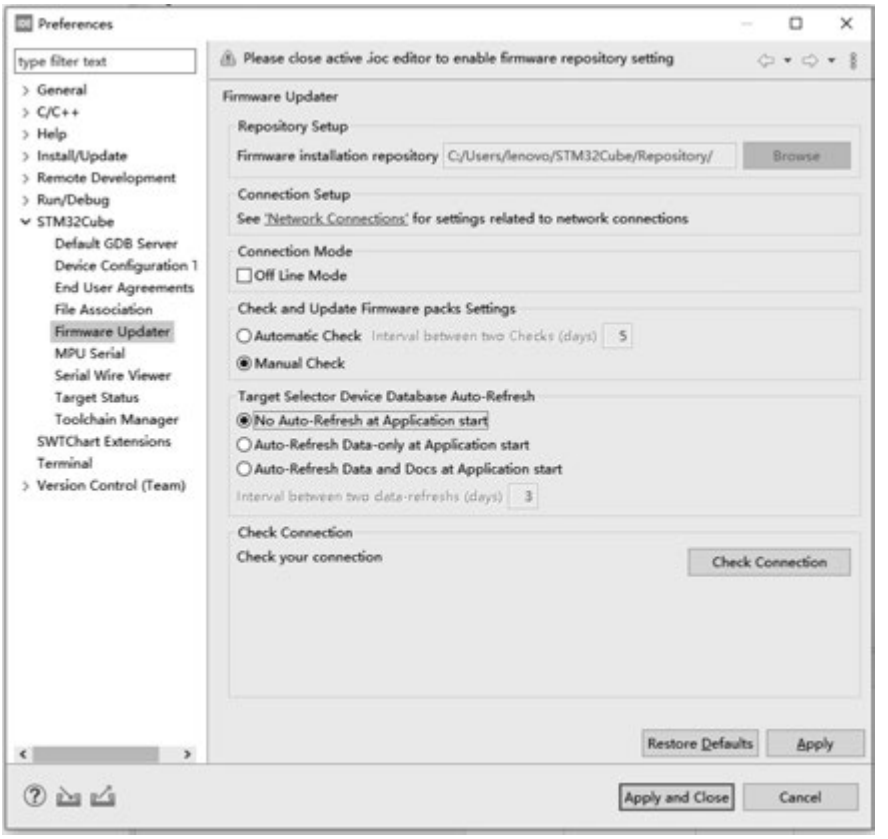

图 3-28　Preferences 对话框

主要的设置内容包括以下几项。

(1) 将软件库目录设置为 C:/Users/lenovo/STM32Cube/Repository/，也就是与独立的 STM32CubeMX 软件共用软件库。

(2) Check and Update Firmware packs Settings 是软件库的更新检查方式，设置为 Manual Check(手动检查)。

(3) Target Selector Device Database Auto-Refresh 是数据和文档的刷新方式，设置为 No Auto-Refresh at Application start(应用程序启动时不自动刷新)。

每次新建一个工作空间时，其 STM32Cube 的软件库位置和升级方式将恢复为默认值。所以，如果要使用内置的 STM32CubeMX 修改 MCU 配置并生成代码，要注意在新建工作空间时重新设置软件库位置和升级方式。

Help 主菜单下有几个与 STM32CubeMX 相关的菜单项，包括以下几项。

(1) Check for Updates(检查更新)：会打开一个对话框，用于检查 STM32Cube MCU 固件包和嵌入式软件包的更新。

（2）Manage Embedded Software Packages(管理嵌入式软件包)：执行 Help→Manage Embedded Software Packages 菜单命令，如图 3-29 所示。

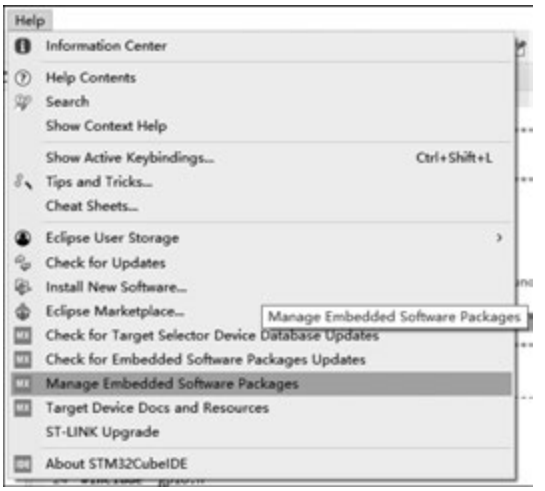

图 3-29 Help→Manage Embedded Software Packages 菜单命令

弹出如图 3-30 所示的 Embedded Software Packages Manager(嵌入式软件包管理器)对话框，对 MCU 固件包和其他嵌入式软件包进行管理。这个对话框的功能与独立的 STM32CubeMX 软件中的嵌入式软件包管理器的功能相同，这里不再赘述。

图 3-30 Embedded Software Packages Manager 对话框

3.3 C/C++场景的界面功能和操作

STM32CubeIDE是基于Eclipse的集成开发环境，专为开发STM32微控制器而设计。它提供了多种功能，帮助用户快速进行嵌入式系统开发。STM32CubeIDE的C/C++场景的一些主要界面功能和操作如下。

1）项目创建和管理

（1）新建项目：通过执行File→New→STM32 Project菜单命令创建新项目，选择目标芯片或开发板，并进行基本设置。

（2）项目资源管理器：在Project Explorer视图中浏览和管理项目文件、源代码和头文件。

2）编辑器

（1）代码编辑：提供语法高亮、错误和警告提示、代码自动补全等功能。

（2）跳转功能：按住Ctrl键并单击函数名或变量名，可以快速跳转到定义或声明处。

（3）注释/取消注释：使用Ctrl+/快捷键注释或取消注释选定的代码行。

3）调试功能

第10集
微课视频

（1）启动调试：通过执行Run→Debug菜单命令或单击工具栏上的调试按钮，启动调试会话。

（2）断点管理：在代码行旁边单击即可添加或删除断点，断点在调试时会使程序暂停执行，以便检查状态。

（3）监视窗口：在调试过程中，Variables视图可显示当前变量的值，Watch视图可手动添加或观察变量。

（4）单步执行：使用工具栏上的Step Into(F5)、Step Over(F6)、Step Return(F7)按钮分别进行逐语句、跳过以及返回上一级调用的调试。

4）代码生成和配置

（1）STM32CubeMX配置：集成STM32CubeMX，可以通过图形界面配置芯片的外设和中间件，然后自动生成初始化代码。

（2）代码生成：STM32CubeMX会根据配置生成HAL(硬件抽象层)库函数调用代码，帮助用户快速开始开发。

5）终端和控制台

（1）串口终端：通过执行Window→Show View→Other→Terminal菜单命令打开串口终端，用于对接串口调试信息。

（2）控制台窗口：显示编译信息和调试输出等日志信息。

6）视图和窗口

（1）Outline视图：显示当前文件的结构，如函数、变量等，帮助快速导航。

（2）Problems视图：列出编译过程中发现的错误和警告信息。

(3) Project Explorer 视图：管理和浏览项目内的文件和文件夹结构。

7) 工具和设置

(1) 编译选项：通过项目属性(执行 Project→Properties 菜单命令)进行编译器、链接器选项和其他构建设置的配置。

(2) 插件扩展：作为 Eclipse 的扩展，STM32CubeIDE 允许安装各种 Eclipse 插件，以增强功能。

一个操作示例如下。

(1) 新建一个 STM32 项目。打开 STM32CubeIDE，执行 File→New→STM32 Project 菜单命令。在 Device Selector 中选择目标芯片(如 STM32F4 系列)，然后单击 Next 按钮。填写项目名称，选择目标语言(C 或 C++)，然后单击 Finish 按钮。

(2) 配置外设和生成代码。打开项目中自动生成的.ioc 文件，会启动 STM32CubeMX 配置界面。在配置界面中进行引脚配置、外设初始化等设置。配置完成后，执行 Project→Generate Code 菜单命令生成初始化代码。

(3) 调试项目。单击工具栏上的 Debug 按钮启动调试。在代码编辑器中设置断点。使用调试视图中的单步执行按钮进行调试。

STM32CubeIDE 集成了丰富的功能，支持从项目创建、代码编写、外设配置到调试的完整开发流程，非常适合 STM32 嵌入式开发。

3.3.1 主要视图

C/C++场景是 STM32CubeIDE 中最常用的一个场景，图 3-20 就是 C/C++场景的典型界面，主要有以下几个视图。

(1) Project Explorer 视图：显示项目目录下的所有文件夹和文件，用于对项目的文件夹和文件进行管理。

(2) 文本编辑器：位于界面中间，用于显示和编辑各种文本文件，主要是程序文件。在 Project Explorer 视图中双击一个文本文件时就可以打开这个文件，并以多页界面显示多个文件。

(3) Outline 视图：显示文本编辑器当前页面中的程序的提纲，如程序中的类型、常量、变量、函数等。在 Outline 视图上单击一个符号时，文本编辑器中就会将输入光标定位到这个符号定义处。Outline 视图中有一个工具栏，工具栏上的按钮可以实现排序、符号过滤等功能。

(4) Console 视图：可以显示编译过程和编译结果信息。

(5) Problems 视图：可以显示编译过程中出现的警告、错误等信息，双击一个提示信息就可以在文本编辑器中定位到出错误的代码行。

(6) Build Analyzer 视图：可以显示项目构建后 Flash、RAM 等存储空间的使用情况。

(7) 还有其他一些有用的视图，如 Tasks 视图。

3.3.2 工具栏功能

STM32CubeIDE的主工具栏按钮会根据场景的变化而变化，如C/C++场景的主工具栏按钮和Debug场景的差别就较大。主工具栏分为左端和右端两大部分，左端的工具栏是与文件操作、项目构建和界面操作相关的按钮，右端的工具栏是场景切换操作的按钮。

图3-31所示为C/C++主工具栏左端的按钮，这些按钮的简要功能说明如表3-1所示，其中标注了备注序号的见后文的详细说明。工具栏上某些按钮并不适合于C/C++场景，或极少使用，用户可以定制工具栏按钮。执行Window→Perspective→Customize Perspective菜单命令，会弹出一个对话框，可供用户自定义场景，设计更适合自己使用习惯的工具栏。

图3-31 C/C++场景主工具栏左端的按钮

表3-1 C/C++场景主工具栏左端的按钮功能说明

按钮图标	提示文字	快捷键	功能说明	备注
	New	—	出现一个下拉菜单，用于新建各种项目和文件	(1)
	Save	Ctrl+S	保存文本编辑器里当前页面的文件	—
	Save All	Ctrl+Shift+S	保存文本编辑器里所有页面的文件	—
	Manage configurations for the current project	—	选择项目的配置模式，Debug模式或Release模式	(2)
	Build projects Build the active configurations of selected projects	—	构建所选项目的活动配置	(3)
	Build all	Ctrl+B	完全重新构建当前项目	—
	Undo	Ctrl+Z	撤销	
	Redo	Ctrl+Y	重做	
	Skip All Breakpoints	Ctrl+Alt+B	忽略所有断点，在程序调试时有用	—
	Device Configuration Tool Code Generation	—	设备配置工具代码	—
	New C/C++ project	—	出现一个下拉菜单，用于新建C/C++项目	—
	New C/C++ Source Folder	—	出现一个下拉菜单，用于新建源代码文件夹	—

续表

按钮图标	提示文字	快捷键	功能说明	备注
	New C/C++ Source File	—	出现一个下拉菜单，用于新建头文件或源程序文件	—
	New C++ Class	—	新建C++类	—
	Debug As...	—	出现一个下拉菜单，用于启动项目调试	—
	Run As...	—	出现一个下拉菜单，用于启动项目运行	—
	External Tools Run Last Tool	—	启动外部工具 运行最后一个工具	—
	Open Element	Ctrl+Shift+T	打开一个对话框，对元素(Element)进行查找	(4)
	Search	—	打开一个查找对话框，有3种类型的搜索	—
	Toggle Mark Occurences	Alt+Shift+O	切换标记同类项，如一个函数内一个变量所有出现的实例	—
	Toggle Word Wrap	Alt+Shift+Y	切换文本自动换行功能	—
	Toggle Block Selection Mode	Alt+Shift+A	切换文本块选择方式	—
	Show Whitespace Character	—	切换显示空格符号	—
	Next Annotation	Ctrl+.	移动到下一标注处，标注包括书签、断点、编译错误、编译警告等。有一个下拉菜单可以选择标注类型	(5)
	Previous Annotation	Ctrl+,	移动到上一标注处	(5)
	Last Edit Location Previous Edit Location	Ctrl+Q	定位到最后一次修改的代码处	—
	Next Edit Location	Ctrl+Alt+Right	下一页编辑位置	
	Back to	Alt+Left	代码追踪时回到上一级	—
	Forward to Forward	Alt+ Right	代码追踪时回到下一级	—
	Pin Editor	—	在文本编辑器当前页面设置或取消图钉标记	—
	Information Center	—	显示信息中心页面	—

表 3-1 中的几个按钮下拉菜单介绍如下。

(1) New 按钮的下拉菜单如图 3-32 所示。其中,STM32 Project 菜单项用于打开一个向导,创建基于 STM32 MCU/MPU 的嵌入式项目;STM32 Project from an Existing STM32CubeMX Configuration File(.ioc)菜单项用于从一个已有的 STM32CubeMX 文件创建项目。

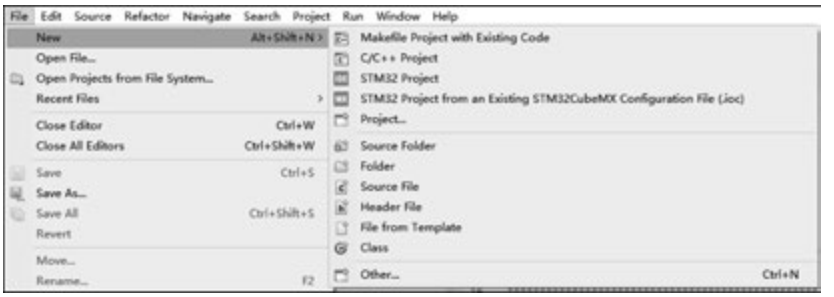

图 3-32 New 按钮的下拉菜单

(2) 一个项目有 Debug 和 Release 两种配置,Manage Configurations for the current project 按钮用于设置项目当前的配置。若单击按钮右侧的箭头图标,会出现 Debug 和 Release 两个菜单项,单击菜单项就可以选择。若直接单击按钮,会弹出如图 3-33 所示的对话框,用于设置项目的当前配置。

(3) 如果直接单击 按钮,会构建项目的当前配置版本。这个按钮也有 Debug 和 Release 两个菜单项,可以选择构建其中某个版本,如图 3-34 所示。

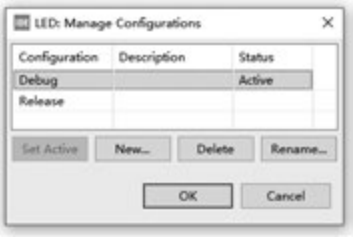

图 3-33 设置项目的当前配置

图 3-34 构建所选项目的活动配置

(4) 单击 Open Element 按钮,会弹出 Open Element 对话框,如图 3-35 所示。可以在整个项目源代码和驱动库源代码中按关键词搜索,并且限定元素类型,如函数、结构体、类、宏定义等。例如,在文本框中输入 GPIO,可以找到 GPIO_InitTypeDef 结构体,双击匹配结果里的这一项,就可以打开其定义所在的源文件,并定位到这个结构体定义的源代码处。

(5) Next Annotation 和 Previous Annotation 这两个按钮有相同的下拉菜单,如图 3-36 所示。每个菜单项是一个复选项,用于选择标注的类型。

主窗口右端的工具栏显示最近打开过的场景,通过这些 Quick Access 按钮可以快速切换到某个场景。例如,工具栏的按钮可能如图 3-37 所示。Open Perspective 按钮 用于打开对话框选择场景,后面的 3 个按钮是最近打开过的 3 个场景,单击即可切换场景。切换到某个场景后,主窗口工具栏左端的工具栏按钮会相应调整。

图 3-35 Open Element 对话框

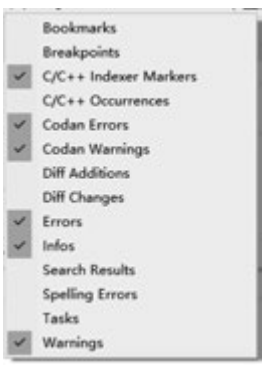

图 3-36 标注类型选择菜单

图 3-37 场景切换工具栏按钮

工具栏上的 按钮用于快速搜索某个功能。例如，单击按钮后输入 Debug，显示的内容如图 3-38 所示，列出了所有包含 Debug 的视图、场景、命令、菜单项等，单击某一项就可以快速执行其功能。

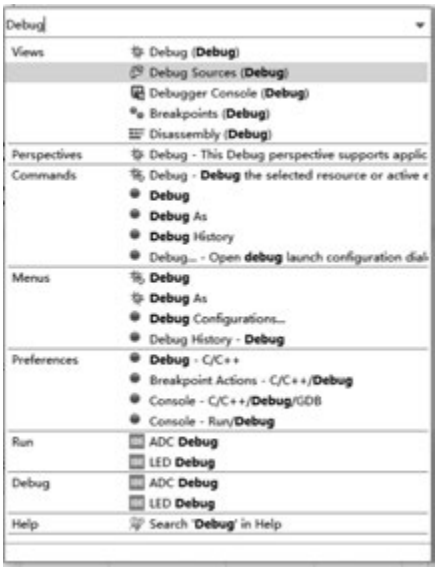

图 3-38　Quick Access 搜索结果

3.3.3　文本编辑器功能和操作

STM32CubeIDE 软件界面中间的区域是文本编辑器，主要用于编辑程序文件。这个编辑器具有强大的编辑功能，下面介绍一些实用的操作功能。

1. 界面主题和编辑器字体

执行 Window→Preferences 菜单命令，会弹出 Preferences 对话框，如图 3-39 所示。在对话框左侧的目录树中单击某个节点，就会在右侧显示具体的设置页面。

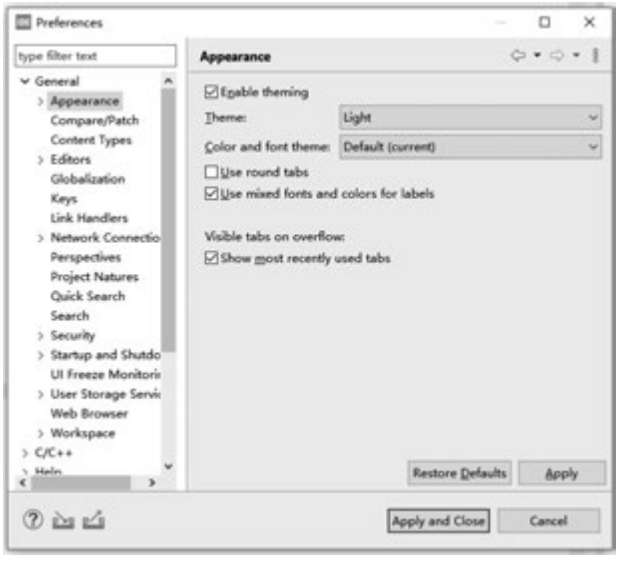

图 3-39　Preferences 对话框

图 3-39 所示为 General→Appearance 设置页面，在这个页面上可以选择软件的界面主题(Theme)、颜色和字体主题(Color and font theme)。单击 Apply 按钮，可立刻应用所设置的选项；单击 Restore Defaults 按钮可以恢复为默认值。

General→Appearance→Color and Fonts 用于设置各种类型文字的字体，如 C 语言程序文件、各种视图界面的字体。颜色和字体设置的操作比较直观和简单，用户自己尝试操作即可，设置页面上都有 Restore Defaults 按钮，可以恢复默认值。

调整编辑器文字的字体大小，可以通过快捷键操作，按住 Ctrl 键和加号键(＋)可以放大字体，按住 Ctrl 键和减号键(－)可以缩小字体。

2. 代码追踪和导航

在编辑器中，当鼠标悬停在某个类型、变量、函数名称等元素上时，会出现一个悬浮框，显示该元素定义的源代码，如图 3-40 所示。按 F2 键或单击此悬浮框，就会弹出为一个具有滚动条的窗口，可以查看完整的代码。这个功能有助于就地查看某个类型、变量、函数等元素的定义，如图 3-41 所示。

图 3-40 元素定义的就地显示(悬浮框)

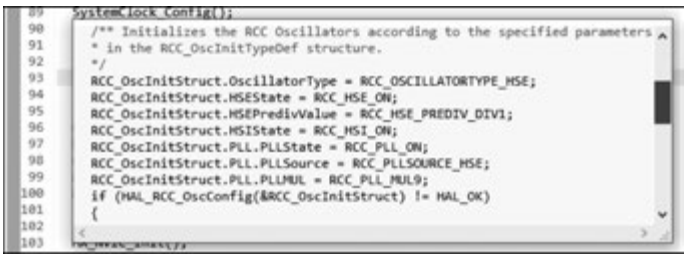

图 3-41 元素定义的就地显示(具有滚动条的窗口)

若要跳转到某个类型、变量、函数名称等元素定义的源代码处，只需将鼠标悬停在这个元素上并按 F3 键，其程序文件就会打开并定位到定义的代码处。

```
95   /* Initialize all configured peripherals */
96   MX_GPIO_Init();
97   MX_USART1_UART_Init();
98   MX_ADC1_Init();
99   MX_FSMC_Init();
100  MX_TIM4_Init();
101
102  /* Initialize interrupts */
103  MX_NVIC_Init();
```

图 3-42　鼠标悬停在 MX_NVIC_Init()函数

使用 F3 键可以在程序文件的函数原型定义和源代码文件的函数代码之间实现跳转。例如，将鼠标悬停在 main.c 文件中的 MX_NVIC_Init()函数的原型定义处（见图 3-42），按 F3 键就会跳转到 main.c 文件中此函数的实现代码处（见图 3-43）。

若只是要在头文件(.h)和程序文件(.c)之间切换，可以使用 Ctrl＋Tab 快捷键。

要在一个文件中快速定位到某个元素，使用 Outline 视图是最方便的，如图 3-44 所示。Outline 视图列出了当前编辑文件中的变量、类型、函数等元素，在 Outline 视图中单击这个元素就能在编辑器中定位到这个元素。

```
main.c
176   }
177   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
178   PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
179   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
180   {
181     Error_Handler();
182   }
183 }
184
185⊖ /**
186   * @brief NVIC Configuration.
187   * @retval None
188   */
189⊖ static void MX_NVIC_Init(void)
190 {
191   /* USART1_IRQn interrupt configuration */
192   HAL_NVIC_SetPriority(USART1_IRQn, 3, 3);
193   HAL_NVIC_EnableIRQ(USART1_IRQn);
194   /* TIM4_IRQn interrupt configuration */
195   HAL_NVIC_SetPriority(TIM4_IRQn, 3, 3);
196   HAL_NVIC_EnableIRQ(TIM4_IRQn);
197 }
```

图 3-43　跳转到 MX_NVIC_Init()函数的实现代码

图 3-44　Outline 视图

3. 编辑功能快捷操作

编辑器的一些有用的快捷操作总结如表 3-2 所示。这些快捷操作在编辑器的快捷菜单或 Edit 和 Source 主菜单的下拉菜单里可以找到。注意，表 3-2 中没有最常用的撤销、剪切、复制等编辑操作。

表 3-2　编辑器的快捷操作

操　作	快　捷　键	功 能 说 明
Toggle Comment	Ctrl＋7 或 Ctrl＋/	在选中的文本行前面加//使其变为注释，或解除注释
Add Block Comment	Ctrl＋Shift＋/	将选中的代码用/＊　＊/进行块注释
Remove Block Comment	Ctrl＋Shift＋\	移除块注释的注释符号
Shift Left	Shift＋Tab	将选中的代码行向左移动，减少缩进
Correct Indentation	Ctrl＋I	自动修正选中代码行的缩进格式
Format	Ctrl＋Shift＋F	自动调整所选行的格式
Toggle Source/Header	Ctrl＋Tab	在源代码文件和头文件之间切换
Toogle Breakpoint	Ctrl＋Shift＋B	在当前行设置或取消断点

3.4 STM32CubeIDE 项目的文件组成

本节将介绍 STM32CubeIDE 项目的文件组成。

STM32CubeIDE 的一个项目就是一个文件夹，项目名称就是文件夹的名称。如图 3-45 所示，除了根目录下的 LED.ioc 是 STM32CubeMX 的文件，其他都是 STM32CubeIDE 项目的文件夹和文件。

STM32CubeIDE 项目中包含所选型号 MCU 必要的驱动程序，包括 CMSIS（Cortex Microcontroller Software Interface Standard）驱动程序和 HAL 驱动程序，还有用户应用相关的程序文件。这些驱动程序是 STM32CubeMX 根据 MCU 的配置自动从安装的嵌入式软件库中复制过来的，用户应用程序包括所用外设的初始化程序。

图 3-45 所示为 Project Explorer 视图中项目 LED 的文件夹和文件，这个视图显示的就是该项目目录下的所有文件夹和文件。该项目选用的 MCU 是 STM32F407ZG。

图 3-45 Project Explorer 视图中项目 LED 的文件夹和文件

第 11 集 微课视频

3.4.1 CMSIS 驱动程序文件

\Drivers\CMSIS 目录下是 CMSIS 的驱动程序，包括 Cortex 内核的驱动程序和具体 MCU 器件的基础定义头文件，它有两个子目录。

（1）\Drivers\CMSIS\Device\ST\STM32F4xx\Include 目录：这个目录下是具体型号 MCU 的相关定义文件。

① stm32f407xx.h：这是 CMSIS 的 STM32F407xx 系列器件的外设访问头文件，其中包含所有外设的地址映射和数据结构定义，以及外设寄存器的定义，用于访问外设寄存器的宏定义等。

② stm32f4xx.h：这是 STM32F4xx 系列器件的配置文件，它根据项目的编译符号定义，在条件编译中包含具体的 MCU 定义头文件（如 stm32f407xx.h），这个文件中有如下代码段。

```
#if defined(STM32F405xx)
    #include "stm32f405xx.h"
#elif defined(STM32F415xx)
    #include "stm32f415xx.h"
#elif defined(STM32F407xx)
```

```
    #include "stm32f407xx.h"    //本例用到的具体型号 MCU
#elif defined(STM32F417xx)
    #include "stm32f417xx.h"
```

stm32f4xx.h 文件中还有如下条件编译语句，用于确定是否包含 HAL 驱动的基础头文件 stm32f4xx_hal.h。

```
#if defined (USE_HAL_DRIVER)
    #include "stm32f4xx_hal.h"
#endif /* USE_HAL_DRIVER */
```

stm32f4xx.h 文件中用到的条件编译符号 STM32F407xx 和 USE_HAL_DRIVER 是在项目的编译设置中定义的。执行 Project→Properties 菜单命令，弹出项目属性设置 Properties for LED 对话框，在 C/C++ Build→Settings 节点的 Tool Settings 页面，单击 MCU GCC Compiler→Preprocessor，就可以看到这两个预处理符号，如图 3-46 所示。

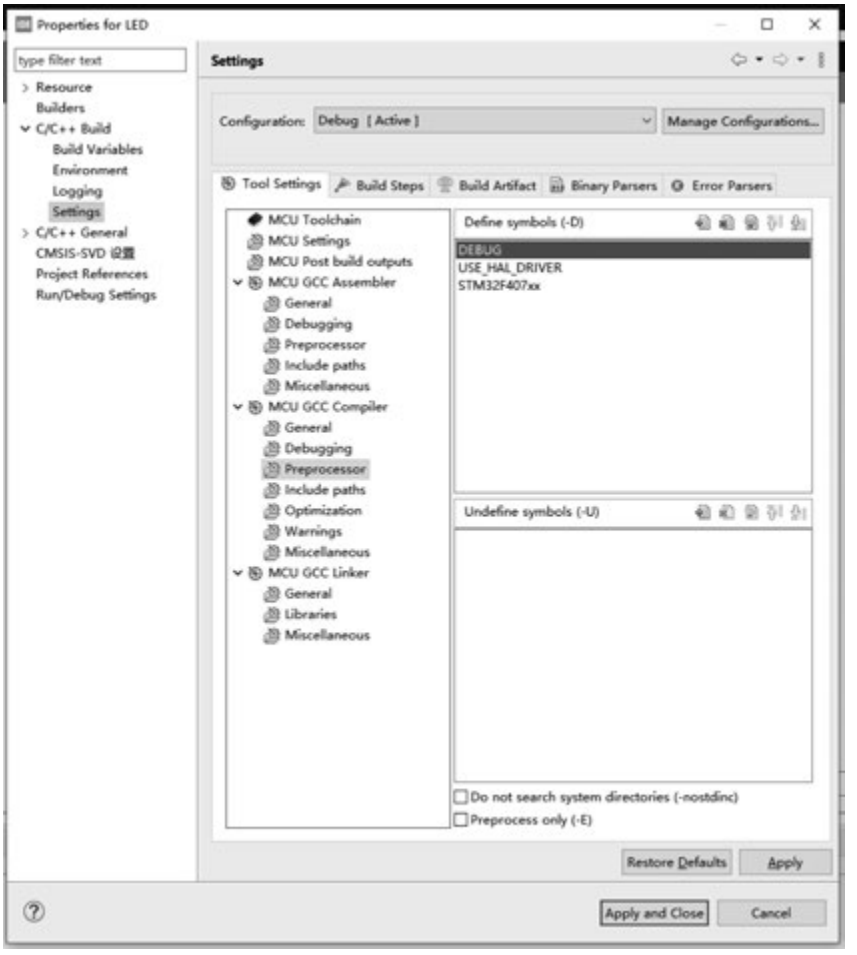

图 3-46　定义预处理符号

③ system_stm32f4xx.h：这个文件里定义了系统初始化函数 SystemInit()，这个函数是在系统复位之后，执行 main()函数之前调用的。SystemInit()函数对 SRAM 的向量表重定位，配置 FSMC/FMC 外设以使用外部的 SRAM 或 SDRAM。

(2) \Drivers\CMSIS\Include 目录：这个目录下都是与 Cortex-M 内核相关的一些文件(见图 3-47)，是 Arm 公司提供的定义文件，与具体的 MCU 型号无关。

图 3-47 Arm 内核驱动文件

3.4.2 HAL 驱动程序文件

STM32 的硬件抽象层(HAL)驱动程序文件是一个官方提供的嵌入式软件库，旨在简化 STM32 微控制器的开发。HAL 驱动程序通过一组 API 函数封装底层硬件操作，提供对微控制器外设(如 GPIO、ADC、UART、SPI 等)的抽象访问，这使得开发者无须深入了解硬件细节即可控制和使用这些外设。此外，HAL 还支持跨平台的代码移植，使得代码能够在不同的 STM32 设备上复用，提高开发效率和代码可维护性。

1. 外设驱动程序

\Drivers\STM32F4xx_HAL_Driver 目录下是 STM32F4xx 系列器件的 HAL 驱动程序。该目录下有两个子目录：\Inc 目录和\Src 目录。\Inc 目录(见图 3-48)下是头文件，\Src 目录下是程序文件。一个头文件对应一个程序文件，分别保存在这两个目录下。每种外设有一个基本驱动文件，有的还有一个扩展驱动文件。

stm32f4xx_hal_ppp.h 是外设 ppp 的基本驱动程序头文件，如 GPIO 的驱动头文件 stm32f4xx_hal_gpio.h、定时器的驱动头文件 stm32f4xx_hal_tim.h 等。

stm32f4xx_hal_ppp_ex.h 是外设 ppp 的扩展驱动程序头文件，包括某个型号或某个系列 MCU 的特定 API 函数，或重新定义的用于替换基本驱动程序的一些 API 函数，如

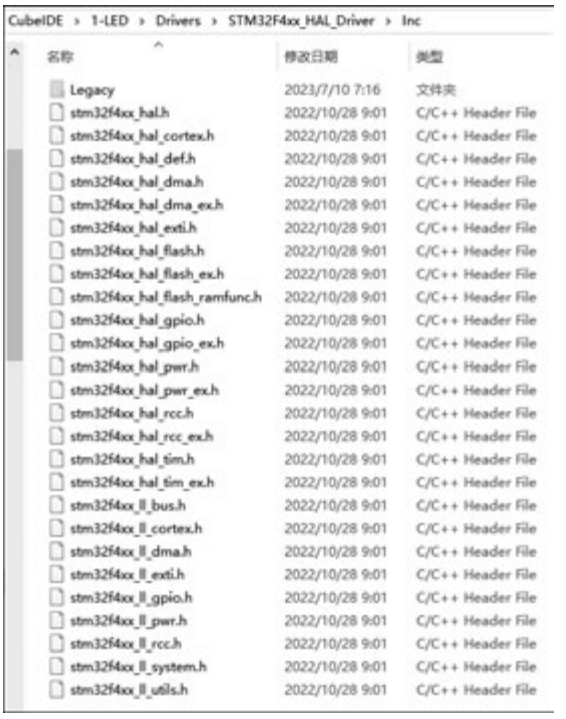

图 3-48 HAL 驱动文件

stm32f4xx_hal_gpio_ex.h、stm32f4xxx_hal_tim_ex.h 等。

\Drivers\STM32F4xx_HAL_Driver 目录下除了外设的驱动程序文件，还有 HAL 库的几个基础文件。

2. HAL 驱动头文件 stm32f4xx_hal.h

stm32f4xx_hal.h 文件中有 HAL 驱动的一些宏定义和函数定义。最主要的一个函数是 HAL_Init()，用于 HAL 库的初始化，其功能是复位所有外设、初始化 Flash 接口、配置系统定时器 SysTick 周期为 1ms。main()函数里首先执行的就是 HAL_Init()函数，代码如下。

```
HAL_StatusTypeDef HAL_Init (void)
{
    /* 配置 Flash Prefetch, Instruction Cache, Data Cache */
    #if (INSTRUCTION_CACHE_ENABLE != 0U)
    _HAL_FLASH_INSTRUCTION_CACHE_ENABLE();
    #endif /* INSTRUCTION_CACHE_ENABLE */
    #if (DATA_CACHE_ENABLE != 0U)
    HAL_FLASH_DATA_CACHE_ENABLE();
    #endif /* DATA_CACHE_ENABLE */
    #if (PREFETCH_ENABLE != 0U)
    _HAL_FLASH_PREFETCH_BUFFER_ENABLE();
    #endif /* PREFETCH_ENABLE */
    /* 设置中断优先级分组策略 */
```

```
    HAL_NVIC_SetPriorityGrouping (NVIC_PRIORITYGROUP_4);
    /* 使用 SysTick 作为基础时钟,配置 SysTick 定时周期为 1ms */
    HAL_InitTick(TICK_INT_PRIORITY);
    /* 初始化底层硬件,与 MCU 相关 */
    HAL_MspInit();
    return HAL_OK;
}
```

HAL_Init()函数中调用的两个函数 HAL_InitTick()和 HAL_MspInit()都是用__weak 修饰符定义的。例如,stm32f4xx_hal.c 文件中 HAL_MspInit()函数的代码如下(保留了英文注释)。

```
__weak void HAL_MspInit (void)
{
    /* NOTE : This function should not be modified, when the callback is needed, the HAL_MspInit
could be implemented in the user file
    注意:这个函数不应该被修改,需要调用时,可以在用户文件里重新实现 HAL_MspInit()函数
    */
}
```

函数前面使用__weak 修饰符,这种函数称为“弱函数”。这个弱函数里没有任何代码,注释表明,如果用户需要在 HAL 初始化时做一些针对 MCU 的初始化操作,可以在自己的程序文件里重新定义这个函数。之后,在项目编译时,编译的就是用户重新实现的 HAL_MspInit()函数,而不是 HAL 库中的弱函数 HAL_MspInit()。在 HAL 库中有大量的弱函数。

stm32f4xx_hal.h 文件还声明了一个常用的延时函数 HAL_Delay(),它基于系统定时器 SysTick 实现精确的毫秒级延时。

3. HAL 通用定义文件 stm32f4xx_hal_def.h

stm32f4xx_hal_def.h 文件中有 HAL 的一些通用定义,包括枚举类型、宏定义、结构体等。例如,HAL 库中很多函数的返回值类型 HAL_StatusTypeDef,就是在这个文件里定义的一个枚举类型,定义如下。

```
typedef enum
{
    HAL_OK          = 0x00U,
    HAL_ERROR       = 0x01U,
    HAL_BUSY        = 0x02U,
    HAL_TIMEOUT     = 0x03U
} HAL_StatusTypeDef;
```

4. Cortex HAL 驱动文件 stm32f4xx _hal_cortex.h

stm32f4xx_hal_cortex.h 文件是 Cortex HAL 驱动文件,它提供 HAL 驱动中用到的 Cortex 内核的一些常量、结构体和函数定义,如中断优先级定义、中断优先级设置函数等。

\Drivers 目录下的驱动程序文件都来自 STM32CubeF4 固件库。在 STM32CubeMX 里,生成 STM32CubeIDE 项目代码时,系统会自动根据 MCU 型号和用到的外设将需要的驱动程序文件复制到 STM32CubeIDE 项目中,并组织好目录结构。\Drivers 目录下的文件

都不要修改，只有极少数情况下需要修改，但是修改后如果用 STM32CubeMX 再次生成代码，所做的修改会丢失。

3.4.3 用户程序文件

用户程序文件架构如图 3-49 所示，在 STM32CubeMX 里设置导出代码选项时，选择生成 .c/.h 文件对。

图 3-49 用户程序文件架构

1. 外设的初始化程序文件

在生成代码时，STM32CubeMX 会为启用的外设都生成一个外设初始化程序文件。例如，在本项目中，使用 PF9 和 PF10 引脚作为输出引脚驱动两个 LED，就用到了 GPIO 外设，所以生成了 GPIO 初始化程序文件 gpio.c 和 gpio.h。gpio.h 文件定义了 GPIO 外设初始化函数原型。

```
void MX_GPIO_Init(void);
```

在 gpio.c 文件中，有 MX_GPIO_Init()函数的实现代码，对用到的两个 GPIO 引脚进行了初始化设置。MX_GPIO_Init()函数的实现代码如下。

```
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* GPIO 端口和时钟使能 */
    HAL_RCC_GPIOF_CLK_ENABLE();
    HAL_RCC_GPIOH_CLK_ENABLE();
    HAL_RCC_GPIOA_CLK_ENABLE();
    /* 配置 GPIO 引脚输出电平，输出 */
    HAL_GPIO_WritePin (LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
    /* 配置 GPIO 引脚输出电平，输出 */
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
    /* 配置 GPIO 引脚：LED1_Pin 和 LED2_Pin 在 main.h 中定义 */
    GPIO_InitStruct.Pin = LED1_Pin|LED2_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOF,&GPIO_InitStruct);
}
```

2. HAL 配置文件 stm32f4xx_hal_conf.h

stm32f4xx_hal_conf.h 文件是对 HAL 驱动程序的一些配置，如启用 MCU 上的哪些外设模块，对 RCC 的 HSE、HSI、LSE、LSI 等时钟频率的设置等。该文件中的部分代码如下。

```
/********** 模块选择 **********/
#define HAL_MODULE_ENABLED
//#define HAL_ADC_MODULE_ENABLED
//#define HAL_CAN_MODULE_ENABLED
```

```
//#define HAL_LPTIM_MODULE_ENABLED
//#define HAL_EXTI_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_EXTI_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
/********** HSE/HSI 频率设置 **********/
#if !defined (HSE_VALUE)
#define HSE_VALUE    ((uint32_t)8000000U)    /* HSE 晶振频率,单位为 Hz */
#endif /* HSE_VALUE */
```

代码段中的模块选择部分未显示完整代码，注释掉了未使用模块的定义语句。代码段中定义了宏 HSE_VALUE 为 8000000，也就是在 STM32CubeMX 的时钟树中设置 HSE 晶振频率为 8MHz。

stm32f4xxhalconf.h 文件的内容是根据 STM32CubeMX 中的配置自动生成的，一般不要直接修改此文件，而是在 STM32CubeMX 里修改配置后重新生成代码。

3. 中断服务例程文件

stm32f4xx_it.h 文件中是中断服务例程(Interrupt Service Routine，ISR)的定义，stm32f4xx_it.c 文件中是 ISR 的实现代码。stm32f4xx_it.h 文件中的一些函数原型定义如下。

```
/**** 导出的函数原型 ****/
void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler (void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);
/* USER CODE BEGIN EFP */
/* USER CODE END EFP */
```

本项目的功能是简单地用 GPIO 引脚驱动 LED，并没有显式地使用中断，但是文件里已经定义了一些 ISR。这些是系统用到的一些中断的 ISR，它们的函数实现代码里一般没有做什么处理，只有 SysTick_Handler()函数有实现代码。

SysTick_Handler()是系统定时器 SysTick 的 ISR，系统定时器 SysTick 每 1ms 中断一次，产生周期为 1ms 的嘀嗒信号。HAL 使用它实现毫秒级精确延时函数 HAL_Delay()。

各种中断的 ISR 名称是固定的，startup_stm32f407zgtx.s 文件中定义了这些 ISR 名称。使用 STM32Cube 开发方式时，我们可以在 STM32CubeMX 里图形化地设置和管理所有中断，生成代码时，会自动在 stm32f4xx_it.h 和 stm32f4xx_it.c 文件中生成已开启中断的 ISR 声明和代码框架，用户一般不需要直接修改 stm32f4xx_it.h 和 stm32f4xx_it.c 文件

的内容。

4. HAL 的 MSP 初始化程序文件

stm32f4xx_hal_msp.c 是 HAL 库的 MSP 程序文件。MSP 是 MCU Specific Package 的缩写,即 MCU 特定程序包。这个文件定义了 HAL 库的 MSP 初始化函数和反初始化(Deinitialization)函数。本项目的这个文件里有 MSP 初始化函数 HAL_MspInit(),代码如下。

```
void HAL_MspInit(void)
{
    /* USER CODE BEGIN MspInit 0 */
    /* USER CODE END MspInit 0 */
    _HAL_RCC_SYSCFG_CLK_ENABLE();
    _HAL_RCC_PWR_CLK_ENABLE();
    /* System interrupt init*/
    /* USER CODE BEGIN MspInit 1 */
    /* USER CODE END MspInit 1 */
}
```

HAL_MspInit()函数的功能是针对具体 MCU 做一些初始化工作。这个函数实际上是对 stm32f4xx_hal.c 文件中用__weak 修饰符定义的弱函数 HAL_MspInit()的重新实现。在主程序里调用 HAL 初始化函数 HAL_Init()时,实际就是调用了在这个文件中重新实现的 HAL_MspInit()函数。

5. 处理器系统初始化文件 system_stm32f4xx.c

system_stm32f4xx.c 文件是\Drivers\CMSIS\Device\ST\STM32F4xx\include 目录下的系统初始化定义头文件 system_stm32f4xx.h 的程序实现文件,主要实现了 SystemInit()和 SystemCoreClockUpdate()两个函数。

SystemInit()函数在系统复位之后、main()函数执行之前执行,其功能是初始化 FPU 设置、向量表重定位、外部存储器配置。这个文件的代码是由 STM32CubeMX 根据配置自动生成的,不需要手动修改。

6. 最小系统调用文件

syscalls.c 和 sysmem.c 文件是 STM32CubeIDE 最小系统需要用到的文件。syscalls.c 文件定义了一些底层函数,sysmem.c 文件定义了内存管理函数。这两个文件里的函数是被 STM32CubeIDE 调用的,用户程序不会直接用到。

7. 主程序文件

主程序文件就是 main.h 和 main.c。main.h 文件的完整代码如下。为使代码更容易阅读,这里剔除了程序中的注释和条件编译不成立的部分。

```
/*文件:main.h */
/* Includes ------------------------------------ */
#include "stm32f4xx_hal.h"
/* Exported functions prototypes ---------------------- */
void Error_Handler(void);
/* Private defines ------------------------------------ */
```

```
#define LED1_RED_Pin GPIO_PIN_6
#define LED1_RED_GPIO_Port GPIOF
#define LED2_GREEN_Pin GPIO_PIN_7
#define LED2_GREEN_GPIO_Port GPIOF
#define LED3_BLUE_Pin GPIO_PIN_8
#define LED3_BLUE_GPIO_Port GPIOF
```

main.h 文件定义了几个宏，这是因为在 STM32CubeMX 中设置了 PF6 引脚的用户标签为 LED1，PF7 引脚的用户标签为 LED2，PF8 引脚的用户标签为 LED3。STM32CubeMX 生成代码时自动在 main.h 文件中为这些定义了用户标签的引脚创建宏定义。

main.c 文件的完整代码如下。为使代码更容易阅读，这里剔除了程序中的注释和条件编译不成立的部分，并且将部分关键注释译为中文。

```
/文件:main.c */
#include "main.h"
#include "gpio.h"
/* Private includes ---------------------------------- */
/* USER CODE BEGIN Includes */
#include "bsp_led.h"
/* USER CODE END Includes */
/* 私有函数原型 */
void SystemClock_Config(void);
int main(void)
{
    /* MCU Configuration---------------------------------- */
    HAL_Init();         //HAL 初始化，包括 MCU 配置，复位所有外设，初始化 Flash 接口和 SysTick
    SystemClock_Config();                   //系统时钟配置
    MX_GPIO_Init();                         //GPIO 初始化
    /* USER CODE BEGIN 2 */
    /* LED 端口初始化 */
    LED_GPIO_Config();
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        LED1( ON );                         //亮
        HAL_Delay(1000);
        LED1( OFF );                        //灭
        HAL_Delay(1000);

        LED2( ON );                         //亮
        HAL_Delay(1000);
        LED2( OFF );                        //灭

        LED3( ON );                         //亮
        HAL_Delay(1000);
        LED3( OFF );                        //灭

        /* 轮流显示 红绿蓝黄紫青白 颜色 */
        LED_RED;
        HAL_Delay(1000);
```

```
            LED_GREEN;
            HAL_Delay(1000);

            LED_BLUE;
            HAL_Delay(1000);

            LED_YELLOW;
            HAL_Delay(1000);

            LED_PURPLE;
            HAL_Delay(1000);

            LED_CYAN;
            HAL_Delay(1000);

            LED_WHITE;
            HAL_Delay(1000);

            LED_RGBOFF;
            HAL_Delay(1000);
            /* USER CODE END WHILE */

            /* USER CODE BEGIN 3 */
        }
        /* USER CODE END 3 */
    }

    void SystemClock_Config(void)
    {
        RCC_OscInitTypeDef RCC_OscInitStruct = {0};
        RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

        /* 配置主要内部调节器的输出电压 */
        __HAL_RCC_PWR_CLK_ENABLE();
        __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

        /* 根据 RCC_OscInitTypeDef 结构中指定的参数初始化 RCC 振荡器 */
        RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPEHSE;      //使用 HSE
        RCC_OscInitStruct.HSEState = RCC_HSE_ON;                       //开启 HSE
        RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;                   //启动主锁相环
        RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;           //PLL 时钟源设置为 HSE
        RCC_OscInitStruct.PLL.PLLM = 25;                               //PLLM 分频器系数 = 25
        RCC_OscInitStruct.PLL.PLLN = 336;                              //PLLN 倍频器系数 = 336
        RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;                    //PLLP 分频器系数 = 2
        RCC_OscInitStruct.PLL.PLLQ = 4;                                //LLQ 分频器系数 = 4
        if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
        {
            Error_Handler();
        }

        /* Initializes the CPU, AHB and APB buses clocks */
        RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
        RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
    {
        Error_Handler();
    }
}

/**
  * @brief This function is executed in case of error occurrence.
  * @retval None
  *错误处理函数*/
void Error_Handler(void)
{
    /* 用户代码开始错误处理调试 */
    /* 用户可以添加自己的代码报告 HAL 错误返回状态 */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}
```

main.c 文件定义了一个 SystemClock_Config()函数，用于系统时钟配置，包括 CPU、AHB 和 APB 总线时钟频率的定义。这个函数的代码是根据 STM32CubeMX 里 RCC 和时钟树的设置自动生成的。因此，若需要修改某个时钟信号频率，应该在 STM32CubeMX 里修改时钟树后重新生成代码。

再看主函数 main()的代码功能，它依次调用了以下 3 个函数，然后进入 while 循环。

(1) HAL_Init()，这是在 stm32f4xx_hal.h 文件中定义的 HAL 库初始化函数。HAL_Init()函数又调用了重新实现的 MSP 函数 HAL_MspInit()，用于特定 MCU 的初始化。

(2) SystemClock_Config()，这是在 main.c 文件中定义的函数，用于对系统时钟进行配置。

(3) MX_GPIO_Init()，这是 GPIO 外设的初始化函数，也就是对 3 个 LED 引脚的 GPIO 初始化，定义在 gpio.h 文件中。

在 STM32CubeIDE 文本编辑器中，因为条件编译的条件不成立而不会被编译的代码会以灰色底色显示，这样容易看出条件编译的影响代码范围。

3.4.4 启动文件

\startup 目录下的 startup_stm32f407zgtx.s 文件是处理器的启动文件，这是一个汇编语言程序文件，是 MCU 复位后首先执行的程序。启动文件的作用是初始化堆栈指针 SP 和代码指针 PC，设置中断程序向量表，执行 system_stm32f4xx.c 文件中的 SystemInit()函数，然后执行主函数 main()。

所有中断的 ISR 名称都在 startup_stm32f407zgtx.s 文件中定义。例如，下面是系统中断和部分可屏蔽中断 ISR 名称的定义。

```
g_pfnVectors:
.word    _estack
.word    Reset_Handler
.word    NMI_Handler
.word    HardFault_Handler
.word    MemManage_Handler
.word    BusFault_Handler
.word    UsageFault_Handler
.word    0
.word    0
.word    0
.word    0
.word    SVC_Handler
.word    DebugMon_Handler
.word    0
.word    PendSV_Handler
.word    SysTick_Handler

/* External Interrupts */
.word    WWDG_IRQHandler                 /* Window WatchDog */
.word    PVD_IRQHandler                  /* PVD through EXTI Line detection */
.word    TAMP_STAMP_IRQHandler           /* Tamper and TimeStamps through the EXTI line */
.word    RTC_WKUP_IRQHandler             /* RTC Wakeup through the EXTI line */
.word    FLASH_IRQHandler                /* Flash */
.word    RCC_IRQHandler                  /* RCC */
.word    EXTI0_IRQHandler                /* EXTI Line0 */
.word    EXTI1_IRQHandler                /* EXTI Line1 */
.word    EXTI2_IRQHandler                /* EXTI Line2 */
.word    EXTI3_IRQHandler                /* EXTI Line3 */
.word    EXTI4_IRQHandler                /* EXTI Line4 */
.word    DMA1_Stream0_IRQHandler         /* DMA1 Stream 0 */
.word    DMA1_Stream1_IRQHandler         /* DMA1 Stream 1 */
.word    DMA1_Stream2_IRQHandler         /* DMA1 Stream 2 */
.word    DMA1_Stream3_IRQHandler         /* DMA1 Stream 3 */
.word    DMA1_Stream4_IRQHandler         /* DMA1 Stream 4 */
.word    DMA1_Stream5_IRQHandler         /* DMA1 Stream 5 */
.word    DMA1_Stream6_IRQHandler         /* DMA1 Stream 6 */
.word    ADC_IRQHandler                  /* ADC1, ADC2 and ADC3s */
.word    CAN1_TX_IRQHandler              /* CAN1 TX */
.word    CAN1_RX0_IRQHandler             /* CAN1 RX0 */
.word    CAN1_RX1_IRQHandler             /* CAN1 RX1 */
.word    CAN1_SCE_IRQHandler             /* CAN1 SCE */
.word    EXTI9_5_IRQHandler              /* External Line[9:5]s */
.word    TIM1_BRK_TIM9_IRQHandler        /* TIM1 Break and TIM9 */
.word    TIM1_UP_TIM10_IRQHandler        /* TIM1 Update and TIM10 */
.word    TIM1_TRG_COM_TIM11_IRQHandler   /* TIM1 Trigger and Commutation and TIM11 */
.word    TIM1_CC_IRQHandler              /* TIM1 Capture Compare */
.word    TIM2_IRQHandler                 /* TIM2 */
.word    TIM3_IRQHandler                 /* TIM3 */
.word    TIM4_IRQHandler                 /* TIM4 */
```

```
.word     I2C1_EV_IRQHandler              /* I2C1 Event */
.word     I2C1_ER_IRQHandler              /* I2C1 Error */
.word     I2C2_EV_IRQHandler              /* I2C2 Event */
.word     I2C2_ER_IRQHandler              /* I2C2 Error */
.word     SPI1_IRQHandler                 /* SPI1 */
.word     SPI2_IRQHandler                 /* SPI2 */
.word     USART1_IRQHandler               /* USART1 */
.word     USART2_IRQHandler               /* USART2 */
.word     USART3_IRQHandler               /* USART3 */
.word     EXTI15_10_IRQHandler            /* External Line[15:10]s */
.word     RTC_Alarm_IRQHandler            /* RTC Alarm (A and B) through EXTI Line */
.word     OTG_FS_WKUP_IRQHandler          /* USB OTG FS Wakeup through EXTI line */
.word     TIM8_BRK_TIM12_IRQHandler       /* TIM8 Break and TIM12 */
.word     TIM8_UP_TIM13_IRQHandler        /* TIM8 Update and TIM13 */
.word     TIM8_TRG_COM_TIM14_IRQHandler   /* TIM8 Trigger and Commutation and TIM14 */
.word     TIM8_CC_IRQHandler              /* TIM8 Capture Compare */
```

在 stm32f4xx_it.h 和 stm32f4xx_it.c 文件中定义和实现 ISR 时，函数名必须与 startup_stm32f407zgtx.s 文件中定义的函数名一致。

3.4.5 根目录下的文件

项目根目录下还有两个扩展名为.ld 的文件，这两个文件是存储器的编译链接脚本文件。

STM32F407ZGTX_FLASH.ld 文件用于设置堆的大小、栈的大小和位置、默认的各种程序段和数据段的地址和长度等。如果使用了外部存储器，还要设置存储器的位置和大小。

STM32F407ZGTX_RAM.ld 文件的功能基本相同，但只是在 RAM 中调试时才用。

这两个文件是根据 STM32CubeMX 中最小堆栈大小的设置，以及 MCU 内部存储空间的分布自动分配而生成的，是比较底层的内容，一般不需详细了解。

如果项目用仿真器下载到开发板上成功调试过，还会在根目录下生成一个扩展名为.launch 的文件，如 LED Debug.launch，这个文件保存了仿真调试器的启动配置参数。

3.4.6 Include 搜索路径

在 Project Explorer 的项目节点下，有一个虚拟的文件夹\Includes，这个文件夹不是硬盘上项目里的实际文件夹，而是项目的 Include 搜索路径。图 3-50 所示为 LED 项目虚拟目录\Includes 下的内容，前 4 条是 STM32CubeIDE 的编译工具链标准库的路径，后 5 条是 LED 项目的 Include 搜索路径。

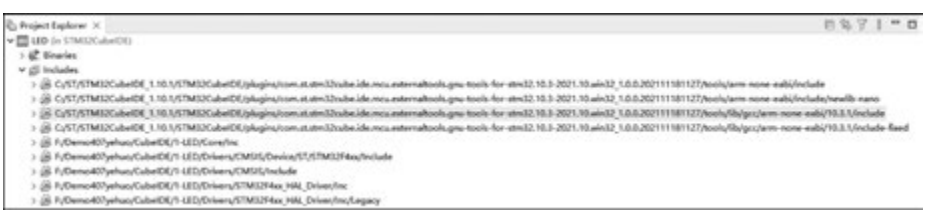

图 3-50 虚拟目录\Includes

3.5 STM32CubeIDE 菜单

STM32CubeIDE 菜单设计合理，提供丰富功能以提升开发效率。File(文件)菜单涵盖新建、打开、保存、导入、导出等文件和项目管理操作。Edit(编辑)菜单提供剪切、复制、粘贴、查找、替换、撤销、重做等编辑功能，提高代码编辑效率。Source(源)菜单包含注释、代码格式化、自动生成代码等工具，便于源代码管理。Refactor(重构)菜单支持代码重命名、提取方法等重构操作，有助于优化代码结构。Navigate(导航)菜单提供快速定位代码、浏览资源、高效导航功能。Search(搜索)菜单允许全局和特定搜索，提升查找效率。Project(项目)菜单支持项目管理和配置，帮助用户高效构建和维护项目。Run(运行)菜单包含启动、调试、断点管理等功能，便于控制程序执行。Window(窗口)菜单允许管理 IDE 布局、视图和透视图。Help(帮助)菜单提供丰富的帮助资源和更新支持，确保获取最新信息和技术支持。整体菜单设计提升了开发、调试和项目管理的效率，为开发者提供一个强大的开发环境。

3.5.1 File(文件)菜单

STM32CubeIDE 的 File(文件)菜单提供了一系列操作选项，方便用户进行文件和项目管理。主要功能包括新建、打开文件和项目、查看最新文件、关闭单个或所有编辑器、保存和另存为、撤销操作、文件移动及重命名、刷新、行分隔符转换(支持 Windows 和 UNIX 格式)、打印、导入、导出、查看属性和切换工作区；此外，还提供了重新启动和退出软件的选项。

图 3-51 File 菜单

File 菜单如图 3-51 所示。

File 菜单功能说明如下。

(1) New：新建。

(2) Open File：打开文件。

(3) Open Projects from File System：从文件系统打开项目。

(4) Recent Files：最近文件。

(5) Close Editor：关闭编辑器。

(6) Close All Editors：关闭所有编辑器。

(7) Save：保存。

(8) Save As：另存为。

(9) Save All：保存所有。

(10) Revert：撤销。

(11) Move：移动。

(12) Rename：重命名。

(13) Refresh：刷新。

(14) Convert Line Delimiters To：转换行分隔符，子菜单如图 3-52 所示。

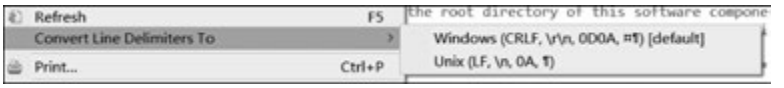

图 3-52 File→Convert Line Delimiters To 子菜单

(15) Print：打印。

(16) Import：导入。

(17) Export：导出。

(18) Properties：属性。

(19) Switch Workspace：切换工作区，子菜单如图 3-53 所示。

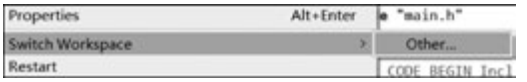

图 3-53 File→Switch Workspace 子菜单

(20) Restart：重新启动。

(21) Exit：退出。

3.5.2 Edit(编辑)菜单

STM32CubeIDE 的 Edit(编辑)菜单提供多种编辑功能以提升代码编辑效率。主要功能包括撤销与重做、剪切、复制、粘贴、删除、全选、扩展选择至特定元素、多选、查找与替换、单词与增量查找、添加书签与任务、智能插入模式、显示工具提示描述、单词自动补全、快速修复、内容辅助和参数提示等；此外，还提供了设置文本编码的选项。

Edit 菜单如图 3-54 所示。

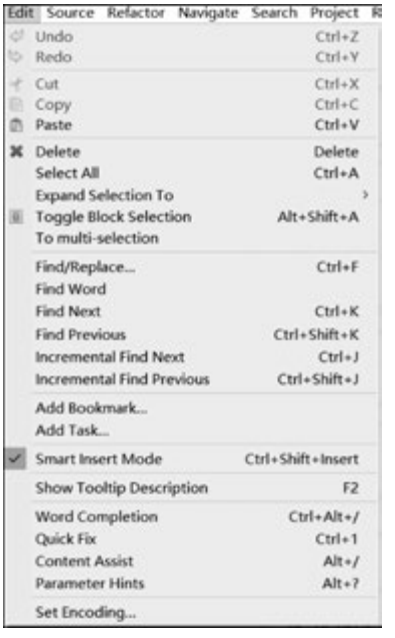

图 3-54 Edit 菜单

Edit 菜单功能说明如下。

(1) Undo：撤销。

(2) Redo：重做。

(3) Cut：剪切。

(4) Copy：复制。

(5) Paste：粘贴。

(6) Delete：删除。

(7) Select All：全选。

(8) Expand Selection To：扩展选择，子菜单如图 3-55 所示。

① Enclosing Element：包围元素。

② Next Element：下一个元素。

③ Previous Element：上一个元素。

④ Restore Last Selection：恢复上次选择。

(9) To multi-selection：到多选。

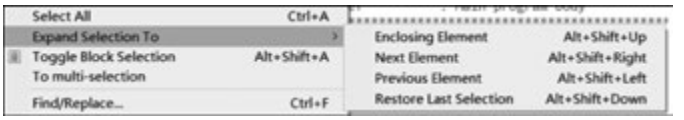

图 3-55 Edit→Expand Selection To 子菜单

(10) Find/Replace：查找/替换。
(11) Find Word：查找单词。
(12) Find Next：查找下一个。
(13) Find Previous：查找上一个。
(14) Incremental Find Next：增量查找下一个。
(15) Incremental Find Previous：增量查找上一个。
(16) Add Bookmark：添加书签。
(17) Add Task：添加任务。
(18) Smart Insert Mode：智能插入模式。
(19) Show Tooltip Description：显示工具提示描述。
(20) Word Completion：单词自动补全。
(21) Quick Fix：快速修复。
(22) Content Assist：内容辅助。
(23) Parameter Hints：参数提示。
(24) Set Encoding：设置编码。

3.5.3 Source(源)菜单

STM32CubeIDE 的 Source(源)菜单提供丰富的源代码管理功能。主要功能包括切换注释、添加和移除块注释、代码行的左右移动、常量对齐、缩进校正、代码格式化、复制限定名称、添加和组织包含文件、行排序、覆盖和实现方法、生成 Getter 和 Setter 方法；此外，还提供包围选中内容的功能，支持配置模板。

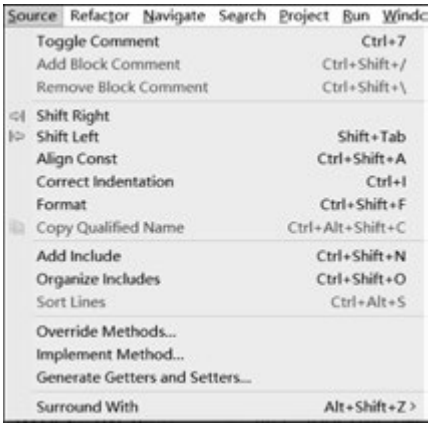

图 3-56 Source 菜单

Source 菜单如图 3-56 所示。

Source 菜单功能说明如下。

(1) Toggle Comment：切换注释。
(2) Add Block Comment：添加块注释。
(3) Remove Block Comment：移除块注释。
(4) Shift Right：向右移动。
(5) Shift Left：向左移动。
(6) Align Const：对齐常量。
(7) Correct Indentation：校正缩进。
(8) Format：格式化。
(9) Copy Qualified Name：复制限定名称。

(10) Add Include：添加包含。

(11) Organize Includes：组织包含。

(12) Sort Lines：排序行。

(13) Override Methods：覆盖方法。

(14) Implement Method：实现方法。

(15) Generate Getters and Setters：生成 Getter 和 Setter 方法。

(16) Surround With：包围选中内容，子菜单如图 3-57 所示。

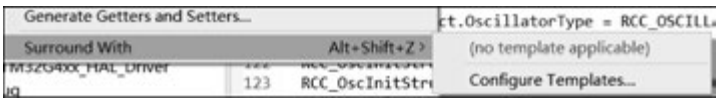

图 3-57 Source→Surround With 子菜单

① (no template applicable)：无适用模板。

② Configure Templates：配置模板。

3.5.4 Refactor(重构)菜单

STM32CubeIDE 的 Refactor(重构)菜单提供多种代码重构功能，以帮助开发者优化代码结构。主要功能包括重命名变量或方法、提取本地变量和常量、提取函数、切换函数显示、隐藏方法、应用和创建脚本以及查看历史变更。这些工具有助于维护代码的可读性和可维护性，提升开发人员在代码优化和重构过程中的效率。

Refactor 菜单如图 3-58 所示。

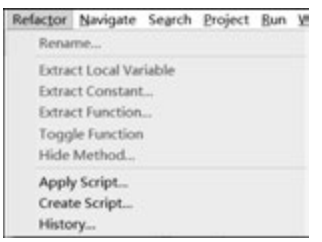

图 3-58 Refactor 菜单

Refactor 菜单功能说明如下。

(1) Rename：重命名。

(2) Extract Local Variable：提取本地变量。

(3) Extract Constant：提取常量。

(4) Extract Function：提取函数。

(5) Toggle Function：切换函数显示。

(6) Hide Method：隐藏方法。

(7) Apply Script：应用脚本。

(8) Create Script：创建脚本。

(9) History：历史。

3.5.5 Navigate(导航)菜单

STM32CubeIDE 的 Navigate(导航)菜单提供丰富的导航功能，帮助开发者高效浏览和定位代码。主要功能包括进入特定目录、转至资源、上一级、匹配括号及书签等位置，还可以打开声明、类型层次结构、调用层次结构和包含浏览器。此外，菜单支持切换源/头文件、快速概要、前后导航及转至行等功能。Show In 子菜单允许在包含浏览器、项目管理器、问题

详情等界面中显示选定资源。这些功能提升了代码浏览和编辑的便利性和效率。

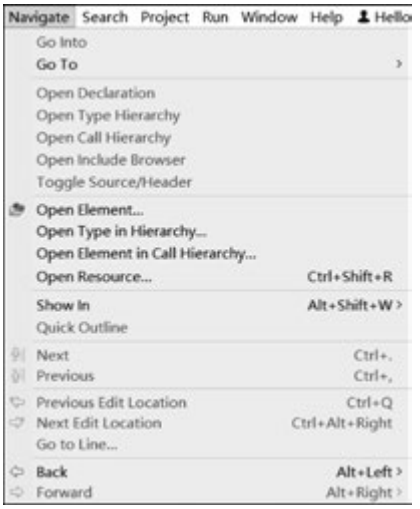

图 3-59　Navigate 菜单

Navigate 菜单如图 3-59 所示。

Navigate 菜单功能说明如下。

(1) Go Into：进入。

(2) Go To：转至，子菜单如图 3-60 所示。

① Back：返回。

② Forward：前进。

③ Up One Level：上一级。

④ Resource：资源。

⑤ Next Member：下一个成员。

⑥ Previous Member：上一个成员。

⑦ Go to Matching Bracket：转到匹配括号。

⑧ Next Bookmark：下一个书签。

(3) Open Declaration：打开声明。

(4) Open Type Hierarchy：打开类型层次结构。

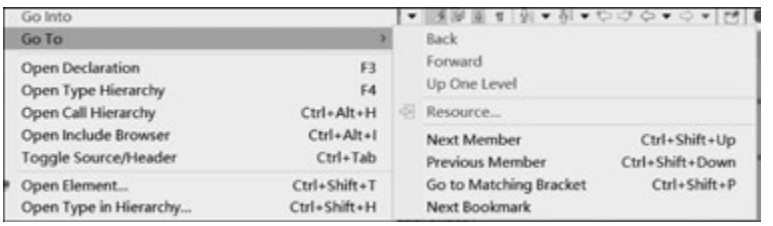

图 3-60　Navigate→Go To 子菜单

(5) Open Call Hierarchy：打开调用层次结构。

(6) Open Include Browser：打开包含浏览器。

(7) Toggle Source/Header：切换源/头文件。

(8) Open Element：打开元素。

(9) Open Type in Hierarchy：在层次结构中打开类型。

(10) Open Element in Call Hierarchy：在调用层次结构中打开元素。

(11) Open Resource：打开资源。

(12) Show In：显示，子菜单如图 3-61 所示。

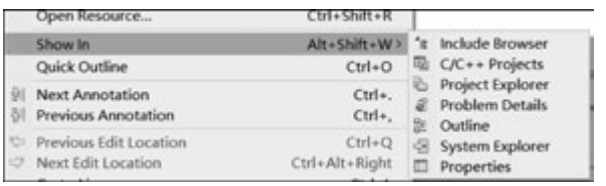

图 3-61　Navigate→Show In 子菜单

① Include Browser：包含浏览器。
② C/C++ Projects：C/C++项目。
③ Project Explorer：项目资源管理器。
④ Problem Details：问题详情。
⑤ Outline：大纲。
⑥ System Explorer：系统资源管理器。
⑦ Properties：属性。
(13) Quick Outline：快速概要。
(14) Next：下一个。
(15) Previous：上一个。
(16) Previous Edit Location：上一个编辑位置。
(17) Next Edit Location：下一个编辑位置。
(18) Go to Line：转至某行。
(19) Back：返回。
(20) Forward：向前。

3.5.6 Search(搜索)菜单

STM32CubeIDE的Search(搜索)菜单提供多种导航和浏览功能，以便开发者高效的在代码和项目中定位。主要功能包括进入特定部分、转至各种资源和代码结构、返回和前进、导航至匹配括号和书签、打开声明和层次结构、切换源文件和头文件、显示在特定视图中、快速概要、定位至编辑位置和行号等。通过这些功能，开发者可以快速在复杂项目中找到所需的代码和资源，提升开发效率。

Search菜单如图3-62所示。

Search菜单功能说明如下。

(1) Search：搜索。
(2) File：文件。
(3) C/C++：搜索项目中的C/C++代码。
(4) Text：文本，子菜单如图3-63所示。

图3-62 Search菜单

图3-63 Search→Text子菜单

① Workspace：工作空间。
② Project：项目。

③ File：文件。

④ Working Set：工作集。

(5) Quick Search：快速搜索。

3.5.7 Project(项目)菜单

STM32CubeIDE 的 Project(项目)菜单提供多种项目管理和构建功能，以便开发者高效管理和编译项目。主要功能包括打开和关闭项目、构建全部或特定项目、配置和管理构建设置、选择和管理工作集、清理项目、自动构建、设置构建目标、管理 C/C++索引等。此外，菜单还提供生成报告和代码、查看和编辑项目属性等功能。这些工具帮助开发者灵活控制项目构建流程，提升生产力和项目管理效率。

Project 菜单如图 3-64 所示。

Project 菜单功能说明如下。

(1) Open Project：打开项目。

(2) Close Project：关闭项目。

(3) Build All：构建全部。

(4) Build Configurations：构建配置，子菜单如图 3-65 所示。

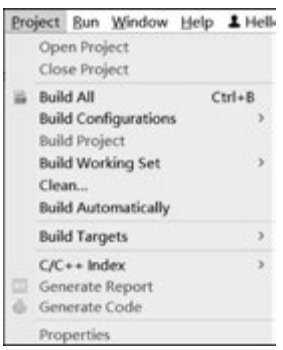

图 3-64 Project 菜单

图 3-65 Project→Build Configurations 子菜单

① Set Active：设置活动。

② Manage：管理。

③ Build by Working Set：按工作集构建。

④ Set Active by Working Set：按工作集设置活动。

⑤ Manage Working Sets：管理工作集。

(5) Build Project：构建项目。

(6) Build Working Set：构建工作集，Select Working Set(选择工作集)子菜单如图 3-66 所示。

(7) Clean：清理。

(8) Build Automatically：自动构建。

(9) Build Targets：构建目标，子菜单如图 3-67 所示。

图 3-66 Project→Build Working Set 子菜单

图 3-67 Project→Build Targets 子菜单

① Create：创建。

② Build：构建。

③ Rebuild Last Target：重新构建上一个目标。

(10) C/C++ Index：C/C++索引，子菜单如图 3-68 所示。

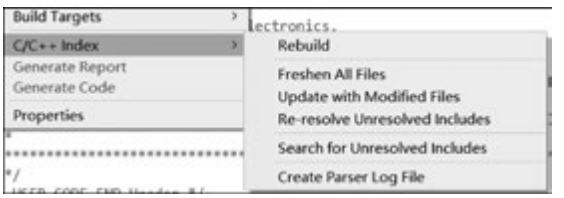

图 3-68 Project→C/C++ Index 子菜单

① Rebuild：重新构建。

② Freshen All Files：刷新所有文件。

③ Update with Modified Files：使用已修改的文件进行更新。

④ Re-resolve Unresolved Includes：重新解析未解析的包含文件。

⑤ Search for Unresolved Includes：搜索未解析的包含文件。

⑥ Create Parser Log File：创建解析器日志文件。

(11) Generate Report：生成报告。

(12) Generate Code：生成代码。

(13) Properties：属性。

3.5.8 Run(运行)菜单

STM32CubeIDE 的 Run(运行)菜单提供了一系列功能，帮助开发者管理和执行项目的运行和调试。主要功能包括运行、调试、查看运行和调试历史、配置运行和调试选项、不同的断点类型(如 C/C++断点和动态打印)和断点管理(如切换、跳过和移除断点)。菜单还包含外部工具的运行和配置选项，有助于扩展 IDE 功能。通过这些功能，开发者可以方便地控制程序的执行流，优化调试和运行效率。

图 3-69 Run 菜单

Run 菜单如图 3-69 所示。

Run 菜单功能说明如下。

(1) Run：运行。

(2) Debug：调试。
(3) Run History：运行历史。
(4) Run As：以…运行。
(5) Run Configurations：运行配置。
(6) Debug History：调试历史。
(7) Debug As：以…调试。
(8) Debug Configurations：调试配置。
(9) Breakpoint Types：断点类型，子菜单如图 3-70 所示。

图 3-70 Run→Breakpoint Types 子菜单

① C/C++ Breakpoints：C/C++断点。
② C/C++ Dynamic Printf：C/C++动态打印。
(10) Toggle Breakpoint：切换断点。
(11) Toggle Line Breakpoint：切换行断点。
(12) Toggle Watchpoint：切换监视点。
(13) Toggle Method Breakpoint：切换方法断点。
(14) Skip All Breakpoints 跳过所有断点。
(15) Remove All Breakpoints：移除所有断点。
(16) External Tools：外部工具，子菜单如图 3-71 所示。

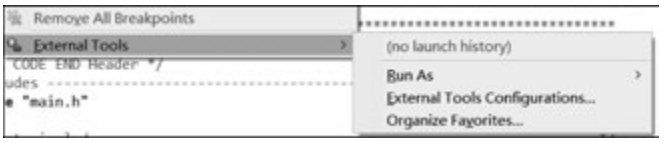

图 3-71 Run→External Tools 子菜单

① (no launch history)：没有启动历史。
② Run As：以…方式运行。
③ External Tools Configurations：外部工具配置。
④ Organize Favorites：组织收藏夹。

3.5.9 Window(窗口)菜单

STM32CubeIDE 的 Window(窗口)菜单提供多种功能以管理视图和编辑器，包括新建独立窗口、编辑器拆分与复制、自动换行、外观设置(如隐藏工具栏与状态栏)、显示不同视图(如控制台和项目资源管理器)以及透视图管理(打开、定制、重置)。这些功能提升了用户的工作效率和界面灵活性。

Window 菜单如图 3-72 所示。

Window 菜单功能说明如下。

(1) New Window：新建窗口。

(2) Editor：编辑器，子菜单如图 3-73 所示。

图 3-72 Window 菜单

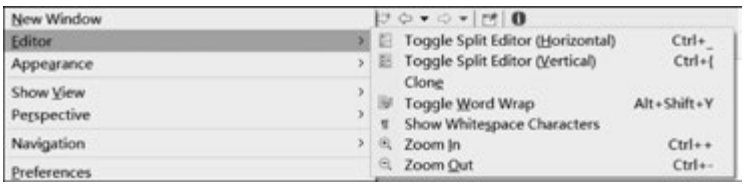

图 3-73 Window→Editor 子菜单

① Toggle Split Editor (Horizontal)：切换拆分编辑器(水平)。

② Toggle Split Editor (Vertical)：切换拆分编辑器(垂直)。

③ Clone：复制。

④ Toggle Word Wrap：切换自动换行。

⑤ Show Whitespace Characters：显示空白字符。

⑥ Zoom In：放大。

⑦ Zoom Out：缩小。

(3) Appearance：外观，子菜单如图 3-74 所示。

图 3-74 Window→Appearance 子菜单

① Hide Toolbar：隐藏工具栏。

② Hide Status Bar：隐藏状态栏。

③ Toggle Full Screen：切换全屏。

④ Maximize Active View or Editor：最大化活动视图或编辑器。

⑤ Minimize Active View or Editor：最小化活动视图或编辑器。

(4) Show View：显示视图，子菜单如图 3-75 所示。

① Build Analyzer：构建分析器。

② Build Targets：构建目标。

③ C/C++ Projects：C/C++项目。

④ Console：控制台。

⑤ Cyclomatic Complexity：圈复杂度。

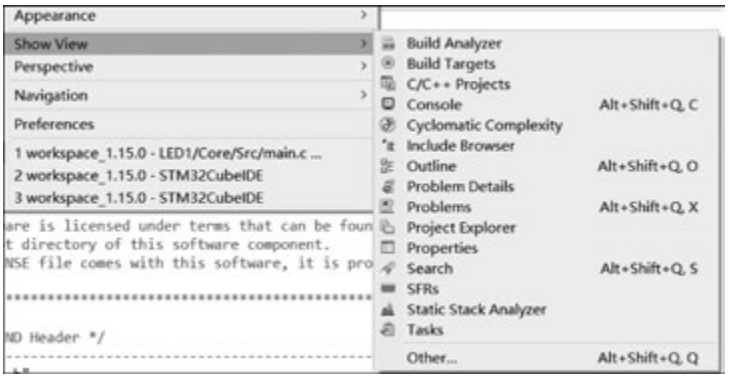

图 3-75 Window→Show View 子菜单

⑥ Include Browser：头文件浏览器。

⑦ Outline：大纲。

⑧ Problem Details：问题详情。

⑨ Problems：问题。

⑩ Project Explorer：项目资源管理器。

⑪ Properties：属性。

⑫ Search 搜索。

⑬ SFRs：特殊功能寄存器。

⑭ Static Stack Analyzer：静态堆栈分析器。

⑮ Tasks：任务。

⑯ Other：其他。

(5) Perspective：透视，子菜单如图 3-76 所示。

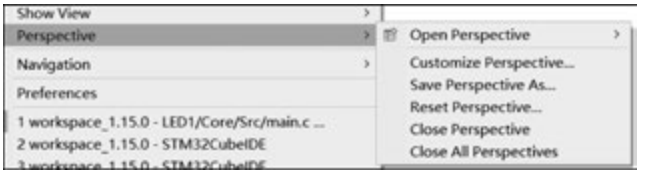

图 3-76 Window→Perspective 子菜单

① Open Perspective：打开透视图。

② Customize Perspective：自定义透视图。

③ Save Perspective As：另存透视图为。

④ Reset Perspective：重置透视图。

⑤ Close Perspective：关闭透视图。

⑥ Close All Perspectives：关闭所有透视图。

(6) Navigation：导航，子菜单如图 3-77 所示。

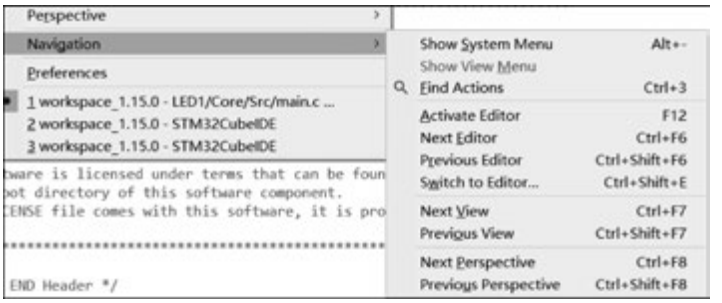

图 3-77 Window→Navigation 子菜单

① Show System Menu：显示系统菜单。
② Show View Menu：显示视图菜单。
③ Find Actions：查找操作。
④ Activate Editor：激活编辑器。
⑤ Next Editor：下一个编辑器。
⑥ Previous Editor：上一个编辑器。
⑦ Switch to Editor：切换到编辑器。
⑧ Next View：下一个视图。
⑨ Previous View：上一个视图。
⑩ Next Perspective：下一个透视图。
⑪ Previous Perspective：上一个透视图。
(7) Preferences：首选项。

3.5.10 Help(帮助)菜单

STM32CubeIDE 的 Help(帮助)菜单提供多种资源和工具，帮助开发者快速获取支持和信息。主要功能包括信息中心、视频教程、帮助内容、搜索、显示上下文帮助和活动键绑定、提示和技巧、技巧表、Eclipse 用户存储管理(如账户、收藏夹和注销)、检查和安装软件更新、访问 Eclipse 市场、检查和管理嵌入式软件包、目标设备文档和资源、ST-LINK 更新以及关于 STM32CubeIDE 的信息。这些功能帮助开发者解决问题、获取最新资源和优化开发过程。

Help 菜单如图 3-78 所示。

Help 菜单功能说明如下。

(1) Information Center：信息中心。
(2) Video tutorials：视频教程。
(3) Help Contents：帮助内容。
(4) Search：搜索。
(5) Show Context Help：显示上下文帮助。
(6) Show Active Keybindings：显示活动键绑定。

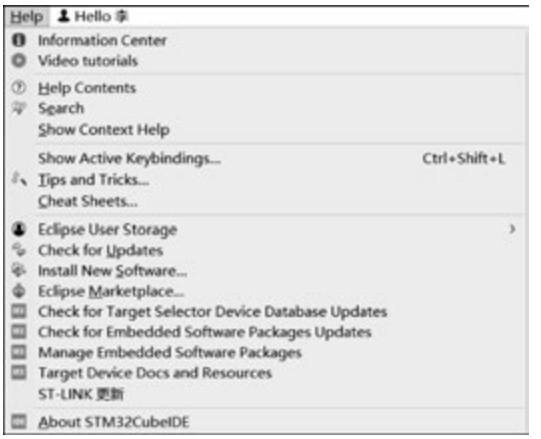

图 3-78 Help 菜单

(7) Tips and Tricks：提示和技巧。

(8) Cheat Sheets：技巧表。

(9) Eclipse User Storage：Eclipse 用户存储，子菜单如图 3-79 所示。

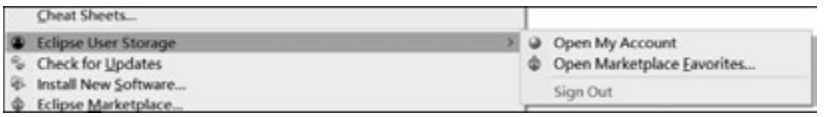

图 3-79 Help→Eclipse User Storage 子菜单

① Open My Account：打开我的账户。

② Open Marketplace Favorites：打开市场收藏夹。

③ Sign Out：注销。

(10) Check for Updates：检查更新。

(11) Install New Software：安装新软件。

(12) Eclipse Marketplace：Eclipse 市场。

(13) Check for Target Selector Device Database Updates：检查目标选择器设备数据库更新。

(14) Check for Embedded Software Packages Updates：检查嵌入式软件包更新。

(15) Manage Embedded Software Packages：管理嵌入式软件包。

(16) Target Device Docs and Resources：目标设备文档和资源。

(17) About STM32CubeIDE：关于 STM32CubeIDE。

3.6 STM32CubeIDE 的操作

本节讲述 STM32CubeIDE 的操作，包括新建和导入工程、项目管理、打开/关闭/删除/切换/导出工程、固件库管理、代码编译、调试及运行配置和启动调试。

3.6.1 新建和导入工程

使用 STM32CubeIDE，可以通过 File 菜单下的 New 或 Import 命令新建或导入一个项目。

也可以执行 File→Open Projects from File System 菜单命令，将一个项目导入当前工作空间中，如图 3-80 所示。

图 3-80 打开 STM32CubeIDE 工程

3.6.2 项目管理

一个工作空间可以管理多个项目，工作空间里的项目有打开和关闭两种状态。图 3-81 所示为 GPIO 输出应用实例 LED 项目工程结构，在项目浏览器里，双击一个项目的节点就可以打开项目，在项目节点上右击，在弹出的快捷菜单中单击 Close Project，就可以关闭这个项目。

当工作空间里有多个项目处于打开状态时，只有一个项目是当前项目，单击一个项目的任何一个文件夹或文件节点，这个项目就变成当前项目。构建、项目属性设置、下载和调试等项目操作都是针对当前项目的。所以，在工作空间中最好只打开一个当前需要处理的项目，关闭其他项目。这样可以减少内存占用，并且可以避免未切换到真正需要处理的项目而导致操作失误。

项目的管理可以通过主工具栏按钮、Project 主菜单或项目浏览器中项目节点的快捷菜单实现。图 3-82 所示为项目节点快捷菜单中的部分菜单项。常用的项目管理操作如下。

(1) Build All(构建全部)：构建工作空间中所有已打开的项目(位于 Project 主菜单下)。所以，不要打开工作空间中不需要处理的项目。

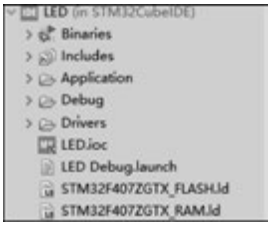

图 3-81　LED 项目工程结构

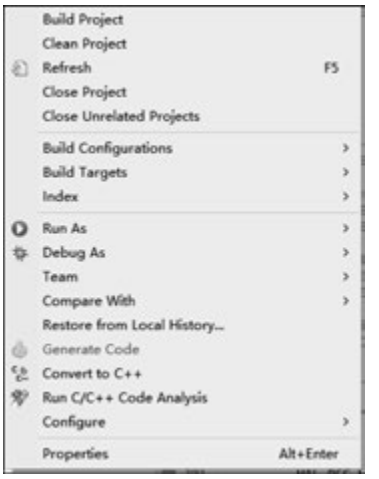

图 3-82　项目节点的快捷菜单(部分项)

(2) Build Project(构建项目)：构建工作空间中的当前项目。构建后会在项目中生成 Debug 或 Release 目录(由项目当前配置决定)和一个虚拟文件夹 Binaries，这个虚拟文件夹中是编译生成的二进制文件，如 LED. elf。

(3) Clean Project(清理项目)：清除项目构建生成的中间文件和二进制文件。

(4) Close Project(关闭项目)：关闭当前项目。

(5) Close Unrelated Projects(关闭不相关项目)：关闭工作空间中所有与本项目无关的项目。

(6) Refresh(刷新)：在使用独立的 STM32CubeMX 重新生成代码后，用户可能需要手动刷新项目的文件。

(7) Build Automatically(自动构建)：这是一个复选项(位于 Project 主菜单下)，如果打开这个选项，在项目程序文件被修改，或 STM32CubeMX 重新生成代码后就会自动构建。一般应关闭此选项，自行控制构建时机。

(8) Properties(属性)：项目属性设置，快捷键为 Alt+Enter，可用于打开一个属性设置对话框，对项目的很多属性进行设置。

3.6.3　打开/关闭/删除/切换/导出工程

在 Project Explorer(工程浏览器)中可以看到当前工作空间下的所有工程，如图 3-83 所示。用户可以对这其中的任一工程进行打开/关闭/删除/切换/导出等操作。STM32CubeIDE 工程浏览器如图 3-83 所示。

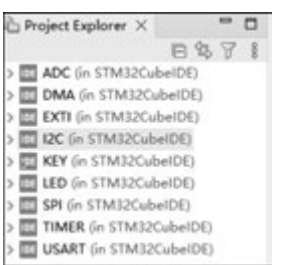

图 3-83　STM32CubeIDE 工程浏览器

3.6.4　固件库管理

STM32CubeIDE 集成了 STM32CubeMX 的部分

功能，可以直接选择芯片/开发板型号，或者选择例程生成一个新工程。STM32CubeIDE 生成工程所需要的驱动和例程代码都来自各 STM32 系列的固件库。

执行 Help→Manage Embedded Software Packages 菜单命令，可以对所有 STM32 固件库以及其他的插件进行管理(安装/删除固件库)。STM32CubeIDE 固件库管理如图 3-84 所示。

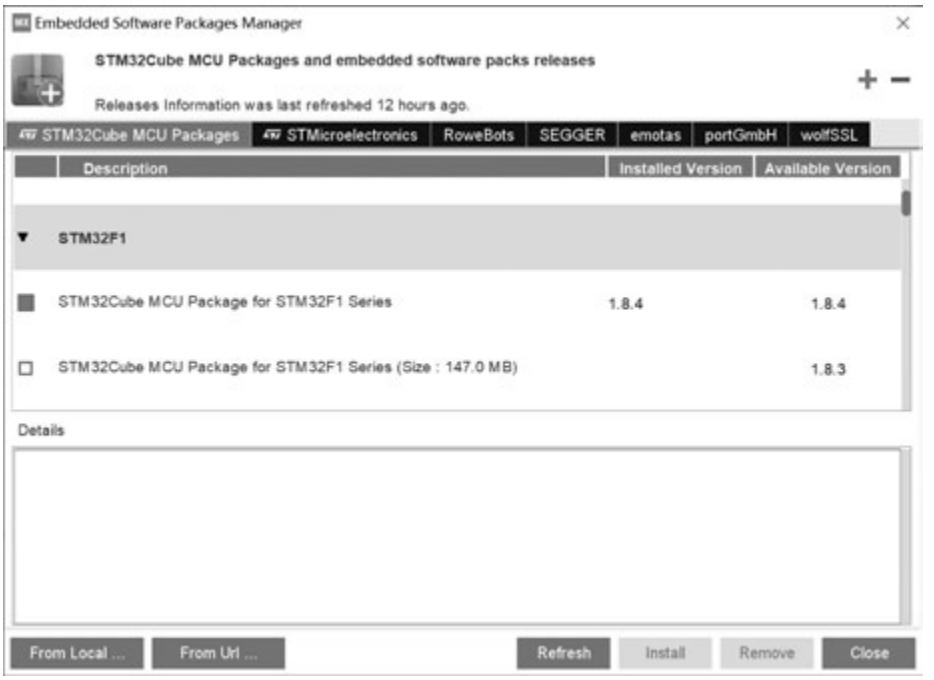

图 3-84 STM32CubeIDE 固件库管理

用户可以通过 Install 按钮让 STM32CubeIDE 自动从网络进行下载安装，也可以通过 From Local 按钮安装已经预先下载好的固件库。通过 Remove 按钮可以删除选中的固件库。

3.6.5 代码编译

用户可以通过以下 3 种方法启动编译。

方法 1：选中工程，右击，从弹出的快捷菜单中选择 Build Project。

方法 2：选中工程，执行 Project→Build Project 菜单命令。

方法 3：选中工程，直接单击工具栏里的 Build 图标。

工程编译完成以后，在 Build Analyzer 窗口可以看到链接文件中定义的所有内存区域(Memory Region)和段(Section)的使用情况，包括加载地址、运行地址、有多少字节已经被占用、还剩余多少字节等。STM32CubeIDE 构建分析结果如图 3-85 所示。

在 Static Stack Analyzer 窗口中显示了静态堆栈的使用情况。STM32CubeIDE 堆栈统计分析如图 3-86 所示。

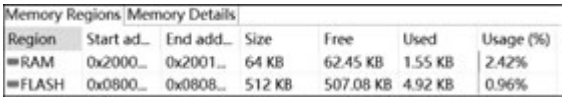

图 3-85　STM32CubeIDE 构建分析结果

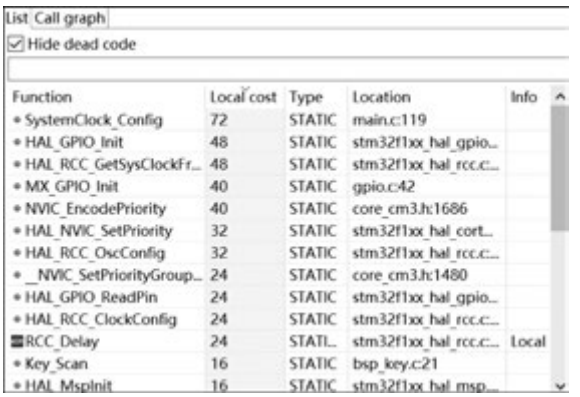

图 3-86　STM32CubeIDE 堆栈统计分析

3.6.6　调试及运行配置

STM32CubeIDE 工程编译完成且无任何错误，就可以进行调试和下载了。

在 C/C++透视图的工具栏中有 3 个和下载调试相关的按钮：调试、运行和外部工具，如图 3-87 所示。

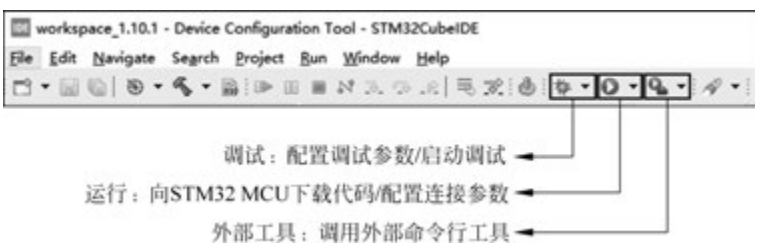

图 3-87　STM32CubeIDE 调试/下载/工具配置说明

通过“调试”按钮旁边的下三角，可以弹出 Debug Configurations 对话框，进行调试参数的配置，如调试器的选择、GDB 连接的设置、ST-LINK 的设置、外部 Flash Loader 的设定等，并启动调试。Debug Configurations 对话框如图 3-88 所示。

通过“运行”按钮，可以仅下载程序，不启动调试。

通过“外部工具”按钮，可以调用外部的命令行工具。

3.6.7　启动调试

在 STM32CubeIDE 中启动调试非常简单。首先，打开项目，确保代码已经编译无误。接着，连接 STM32 开发板到计算机，并在 STM32CubeIDE 中选择相应的调试配置，如

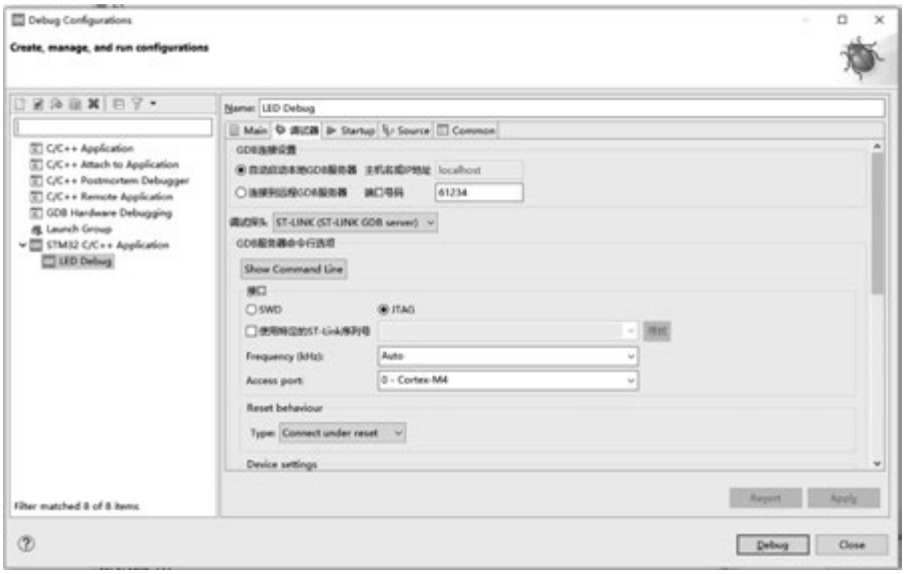

图 3-88 Debug Configurations 对话框

ST-LINK。单击工具栏上的 Debug 按钮，启动调试会话。STM32CubeIDE 会自动下载程序到目标板，并进入调试模式，用户可以设置断点、单步执行代码、观察变量和寄存器等。整个过程直观高效，有助于快速定位和解决问题。

1. 仿真器检测和固件升级

下载程序之前，需要用仿真器连接计算机和开发板，还要在计算机上安装仿真器的驱动程序。STM32CubeIDE 只支持 ST-LINK 仿真器，在首次使用仿真器之前，用户还需要检查仿真器固件版本是否符合 STM32CubeIDE 的要求。方法是执行 Help→“ST-LINK 更新”菜单命令，如图 3-89 所示。

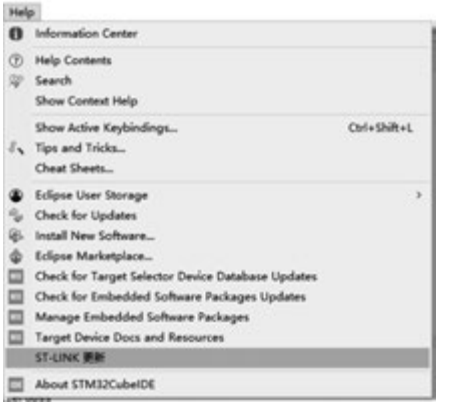

图 3-89 Help→ST-LINK 更新

执行 Help→“ST-LINK 更新”菜单命令后，弹出 STLinkUpgrade 3.4.0 对话框，如图 3-90 所示的。在此对话框中，软件会显示自动发现的仿真器类型，如图 3-90 中下拉列表框里的 ST-LINK/V2。单击 Open in update mode 按钮，如果仿真器与计算机连接正常，会显示仿真器类型和当前版本，以及可以升级到的固件版本。如果当前版本低于可升级版本，就需要升级仿真器的固件，单击 Upgrade 按钮就可以开始升级。仿真器固件升级一次即可，除非有新的固件版本需要升级。特别需要注意的是，ST-LINK/V2 仿真器一定不能连接开发板，待 ST-LINK/V2 仿真器升级完成后，再连接开发板。

用户还可以检测仿真器连接是否正常。有时仿真器虽然连接着计算机，但是无法正常

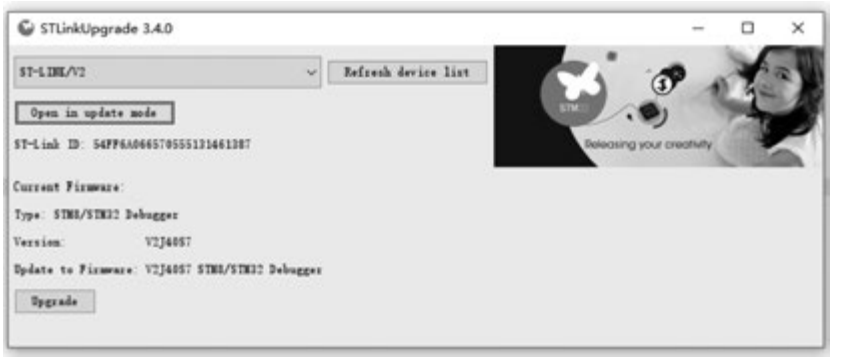

图 3-90　升级仿真器固件

下载程序。打开仿真器固件升级对话框，单击 Open in update mode 按钮，如果软件无法检测到仿真器固件版本，并且在窗口下方显示信息“ST-LINK is not in the DFU mode. Please restart it”，这就说明仿真器连接出现了问题。这种情况下，从 USB 接口拔掉仿真器，然后重新插上就可以了。

STM32CubeIDE 使用 GDB 进行调试，支持 ST-LINK 调试器，支持通过 SWD 或 JTAG 接口连接目标 MCU。

2. 程序下载

构建项目无误后，就可以下载程序到开发板上进行调试了。下载程序之前，可以先在源代码中设置断点，当输入光标在某一有效代码行时，按 Ctrl＋Shift＋B 快捷键就可以在当前行设置或取消断点。

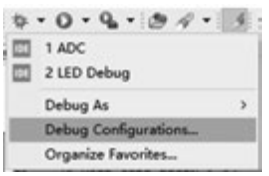

图 3-91　Debug 菜单

单击主工具栏上的“调试”按钮就可以启动程序下载和调试。

单击“调试”按钮 ▾ 右侧的下三角 ▾，弹出如图 3-91 所示的 Debug 菜单，用于调试设置(Debug Configurations)。

单击 Debug Configurations 菜单项，会弹出如图 3-92 所示的 Debug Configurations 对话框。

Debug Configurations 对话框用于设置调试相关的一些参数，所显示的“调试器”标签页的设置都是默认设置。“调试探头”下拉列表用于选择仿真器类型，只有 ST-LINK 和 J-LINK 可选(实际上只能用 ST-LINK)，因为 STM32CubeIDE 只支持这两种仿真器。调试器有 SWD 和 JTAG 两种接口，这个会自动与 STM32CubeMX 里设置的调试接口一致。对于 STM32F407 开发板，我们应该使用 SWD 接口。Reset behaviour 的 Type 下拉列表用于设置下载程序时的复位方式，使用默认的 Connect under reset 即可，而且在调试低功耗程序时，必须使用这种方式。

其他标签页的设置也都保持默认设置即可，然后单击 OK 按钮，会弹出一个提示对话框，提示会切换到 Debug 场景，确认后软件界面就会切换为 Debug 场景。

首次启动调试后，系统会在项目根目录下创建一个扩展名为. launch 的文件，这是记录调试配置的启动文件。下次再单击工具栏上的 Debug 按钮，就可以直接下载程序并进入调

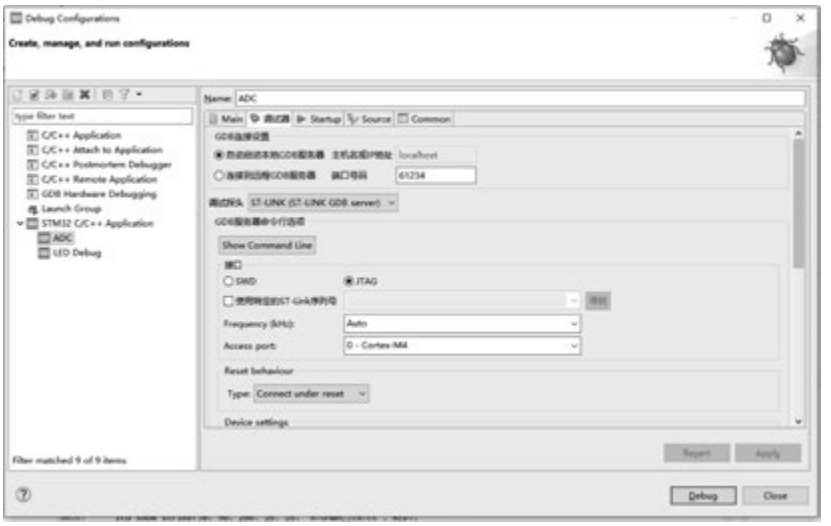

图 3-92 Debug Configurations 对话框

试状态，无须再设置。

STM32CubeIDE 工程编译完成之后，直接单击工具栏的“调试”按钮 或执行 Run→Debug 菜单命令，可以启动调试。

3. Debug 场景界面和调试操作

下载程序后，软件自动切换到 Debug 场景界面，如图 3-93 所示。主工具栏的按钮自动

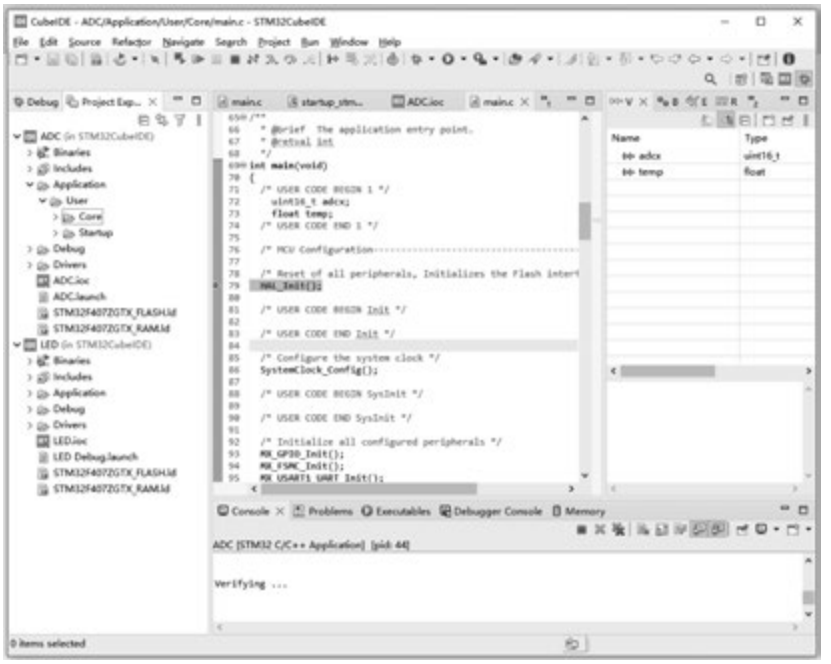

图 3-93 Debug 场景界面

更换了一批，界面上出现了一些调试时用的视图。例如，Variables（变量）视图可以自动显示当前函数内变量的值，Expressions（表达式）视图可以自己添加观察变量，Breakpoints（断点）视图可以添加或移除断点，Registers（寄存器）视图会显示 CPU 寄存器的内容，SFRs（特殊功能寄存器）视图会显示 MCU 上的各种特殊功能寄存器的内容。这些视图的操作就不具体介绍了，用户自己摸索操作一下就可以掌握。

在 Debug 场景里，Run 主菜单包含所有调试操作菜单项，主工具栏上也有一些常用的调试操作按钮，这些按钮的功能如表 3-3 所示。程序单步调试和断点调试的方法与一般的 IDE 软件里的程序调试方法是一样的，这里就不再具体介绍。

表 3-3　用于调试操作的工具栏按钮的功能

图标	提示文字	快捷键	功 能 说 明
	Skip All Breakpoints	Ctrl＋Alt＋B	一个复选按钮，按下后将忽略所有断点
	Terminate and relaunch	—	终止程序，重新下载程序后开始调试
	Resume	F8	从当前停止的行开始，继续连续运行
	Suspend	—	程序挂起
	Terminate	Ctrl＋F2	终止程序执行，并退出 Debug 场景，返回 C/C++场景
	Disconnect	—	断开连接，并退出 Debug 场景，返回 C/C++场景
	Step Into	F5	若当前行是一个函数调用，将会跳转进入此函数的代码内执行
	Step Over	F6	一步执行当前行程序，不管是否有函数调用。如果执行的函数里有断点定义，会执行到断点处暂停
	Step Return	F7	如果当前代码段是跳转进来的一个函数，单击此按钮可以执行完此函数的代码，返回到上一层
	Run to Line	Ctrl＋R	运行到光标所在的行
	Instruction stepping mode	—	进入反汇编（Disassembly）视图，单步调试汇编语言指令
	Reset the chip and restart debug session	—	芯片复位，重新开始调试过程

进入 Debug 场景后，程序会停在 main()函数的第一行代码处。用户可以单步调试，也可以按 F8 快捷键使程序连续运行。

如果是第一次对当前工程进行调试，STM32CubeIDE 会先编译工程，然后打开调试配置窗口。调试配置窗口包含调试接口的选择、ST-LINK 的设置、复位设置和外部 Flash Loader 的设置等选项，用户可以检查或者修改各项配置。确认所有配置都正确无误，就可以单击 OK 按钮，启动调试。

STM32CubeIDE 会先将程序下载到 MCU，然后从链接文件(*.ld)中指定的程序入口开始执行。程序默认从 Reset_Handler 开始执行，并暂停在 main()函数的第一行，等待接下来的调试指令。

启动调试后，STM32CubeIDE 将自动切换到调试透视图，在调试透视图的工具栏中，列出了调试操作按钮，如图 3-94 所示。

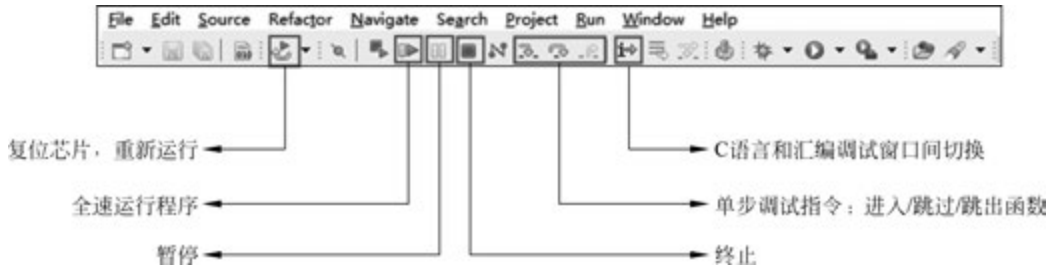

图 3-94 STM32CubeIDE 调试工具栏说明

3.7 使用内置的 STM32CubeMX

STM32CubeIDE 是基于 Eclipse 的软件，Eclipse 可以安装各种插件扩展功能。用户可以在 TrueSTUDIO 中安装 STM32CubeMX 插件，实现在 TrueSTUDIO 中使用 STM32CubeMX。STM32CubeIDE 中集成了 STM32CubeMX，实际上就是预装了 STM32CubeMX 插件。

ADC 项目结合使用了独立的 STM32CubeMX 和 STM32CubeIDE，也可以直接使用 STM32CubeIDE 内置的 STM32CubeMX，其用法基本也是一样的。但是在实际使用中，使用内置的 STM32CubeMX 不如使用独立的 STM32CubeMX 方便，因为使用独立的 STM32CubeMX 软件，界面更大一些，在两个软件之间切换和对比也方便，而且内置的 STM32CubeMX 只能导出 STM32CubeIDE 项目代码，而独立的 STM32CubeMX 可以生成各种 IDE 软件的项目代码。

但是，作为一个功能，下面也介绍一下内置 STM32CubeMX 的使用。要在 STM32CubeIDE 中使用内置的 STM32CubeMX 配置 MCU 和生成代码，也需要配置软件库目录，安装 MCU 固件库。前面介绍了配置软件库目录的方法，与独立的 STM32CubeMX 使用同一个软件库即可。

3.7.1 创建项目

从头开始在 STM32CubeIDE 中创建一个使用 STM32 MCU 的 TESTLED 项目，实现功能与 LED 项目相同。单击 STM32CubeIDE 的“信息中心”按钮 (Information Center)，弹出 STM32CubeIDE 内嵌 STM32CubeMX 界面，几乎与独立的 STM32CubeMX 界面是一样的。

首先在 F 盘创建一个文件夹 DemoLED。

在 STM32CubeIDE 中执行 File→New→STM32 Project 菜单命令，如图 3-95 所示。

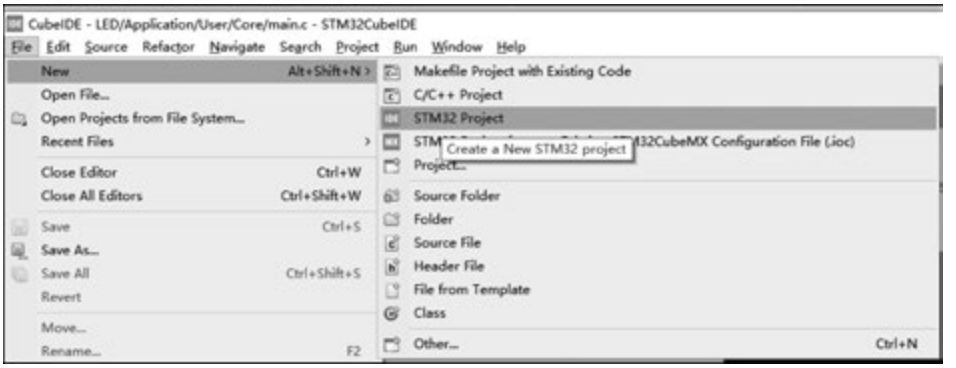

图 3-95　New→STM32 Project 菜单命令

然后，进入如图 3-96 所示的 STM32 Project 对话框，MCU/MPU Selector 标签页用于选择一个 MCU 或 MPU 创建项目。

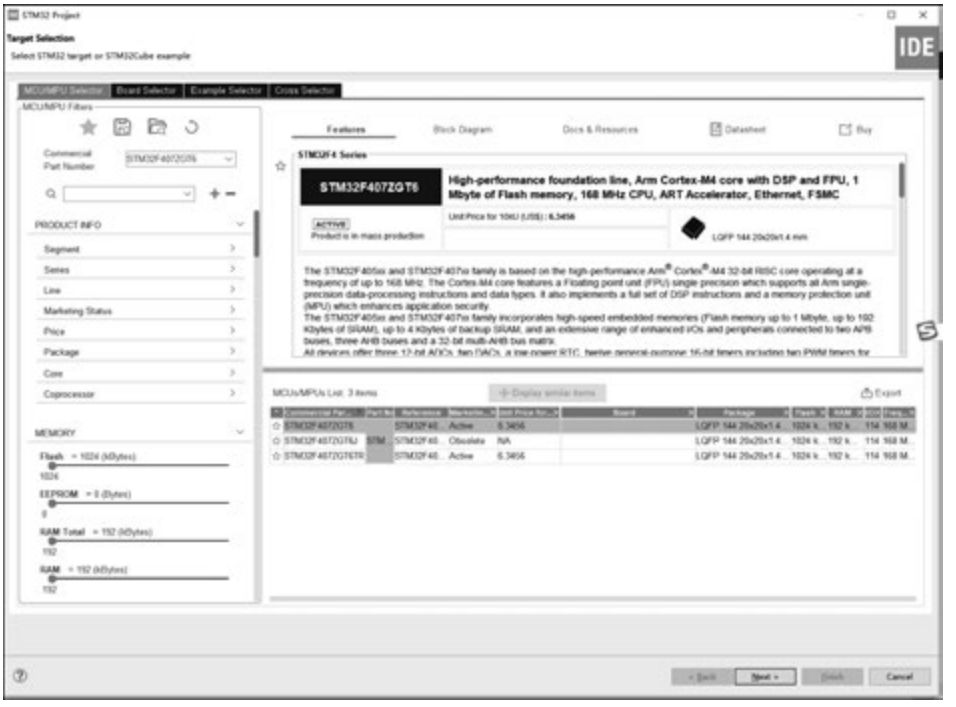

图 3-96　选择 MCU/MPU

选择 STM32F407ZGT6，然后单击 Next 按钮，设置项目名称，如 TESTLED，如图 3-97 所示。如果勾选了 Use default location 复选框，就会在当前工作空间的目录下创建与项目名称同名的子目录。Options 区域有 3 个选项，使用图 3-97 的设置即可，如果需要使用 C++编程，可以选择 C++。

用户可以单击 Finish 按钮直接创建项目，也可以单击 Next 按钮，设置固件版本和生成代码的选项，如图 3-98 所示。

图 3-97 设置项目名称

图 3-98 固件版本和生成代码选项设置

单击 Finish 按钮，将创建 TESTLED 项目，加载 IOC，如图 3-99 所示。

图 3-99　加载 IOC

创建 TESTLED 项目完成，在项目浏览器中打开这个项目，自动切换到 Device Configuration Tool 场景，如图 3-100 所示。

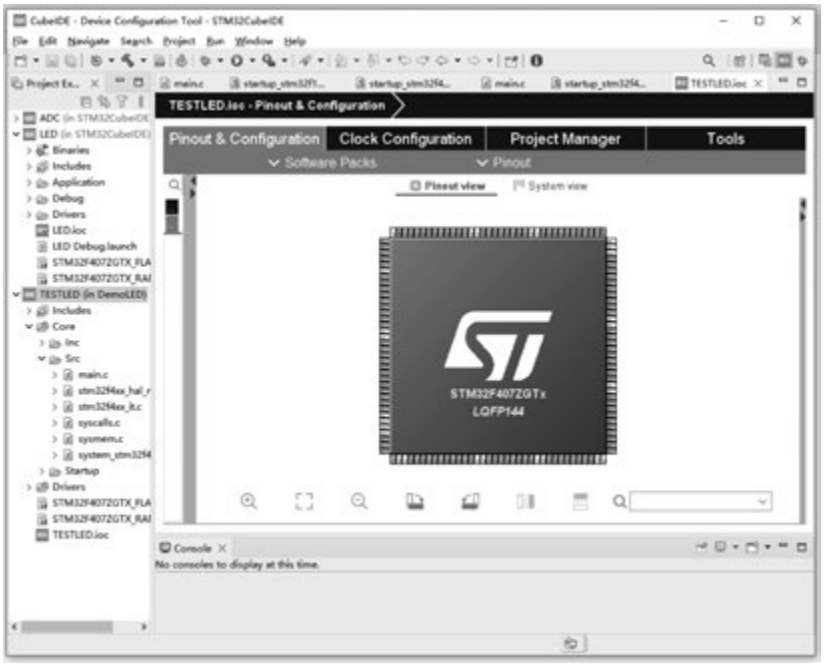

图 3-100　新建项目后的 Device Configuration Tool 场景

3.7.2 配置 MCU 和生成代码

图 3-100 所示的项目浏览器显示了 TESTLED 项目的目录结构。这个项目的根目录下有一个\Core 目录，这个目录下有 3 个子目录\Inc、\Src 和\Startup。

图 3-100 中的工作区是 STM32CubeMX 视图，用于配置 TESTLED. ioc 文件。STM32CubeMX 视图的界面和操作功能与独立的 STM32CubeMX 软件基本相同，只是这里没有生成代码的按钮，项目浏览器里生成项目的 IDE 软件固定为 STM32CubeIDE，不能选择其他 IDE 软件。

此处不再赘述在 STM32CubeMX 视图中进行 MCU 配置的操作，请参考 STM32CubeMX 的内容。本项目 MCU 的具体设置与项目 LED 的完全相同，设置内容如下。

(1) RCC 组件的 HSE 设置为 Crystal/Ceramic Resonator，不启用 LSE。

(2) 在时钟配置页面，将 HSE 设置为 8MHz，选择 HSE 作为主锁存器的时钟源，将 HCLK 设置为 168MHz，由软件自动设置时钟树各种参数。

(3) 在 SYS 组件的模式设置中，将 Debug 接口设置为 Serial Wire。

(4) 在引脚视图上，直接将 PF6、PF7 和 PF8 引脚设置为 GPIO_Output，并修改用户标签为 LED1、LED2 和 LED3。

(5) 在 GPIO 组件的配置页面，对 PF6、PF7 和 PF8 引脚的 GPIO 属性进行设置。

在单击保存文件时，用户会收到是否生成代码的提示，可以选择生成代码，也可以执行 Project→Generate Code 菜单命令(快捷键 Alt+K)或按工具栏上的按钮生成代码。生成代码后再进行构建、下载和调试。

对于使用独立的 STM32CubeMX 软件生成的 LED 项目，在 STM32CubeIDE 的项目浏览器里双击项目 LED. ioc 文件时，也会用 STM32CubeMX 视图打开这个文件，也可以进行配置和生成代码，生成代码时不会破坏项目原来的文件组织结构。

虽然可以使用内置的 STM32CubeMX 在 STM32CubeIDE 里创建 STM32 项目，但是一般多采用独立的 STM32CubeMX 软件。

3.8 STM32CubeIDE 使用偏好设置

STM32CubeIDE 是比较庞大的 IDE 软件，执行 Window→Preferences 菜单命令，弹出 Preferences 对话框，就可以对软件环境进行很多设置。我们之前在这个对话框里设置过软件库目录和更新方式，这里再介绍几个比较有用的设置。

用户可以在 Preferences 对话框左侧目录树上方的文本框里输入搜索关键字。如图 3-101 所示，输入 startup 进行搜索，并单击 Startup and Shutdown 节点。这里设置软件启动时启动的插件，这些插件默认是都启动的。启动的插件越多，软件启动越慢，占用的内存越多。用户可以将一些不常用的插件设置为不启动。例如，图 3-101 中只保留了两个插件启动，这两个插件是构建和调试 STM32 项目必需的。STM32CubeMX 插件也没有设置为启动，在

STM32CubeIDE 里双击一个.ioc 文件时，STM32CubeMX 插件才会启动。

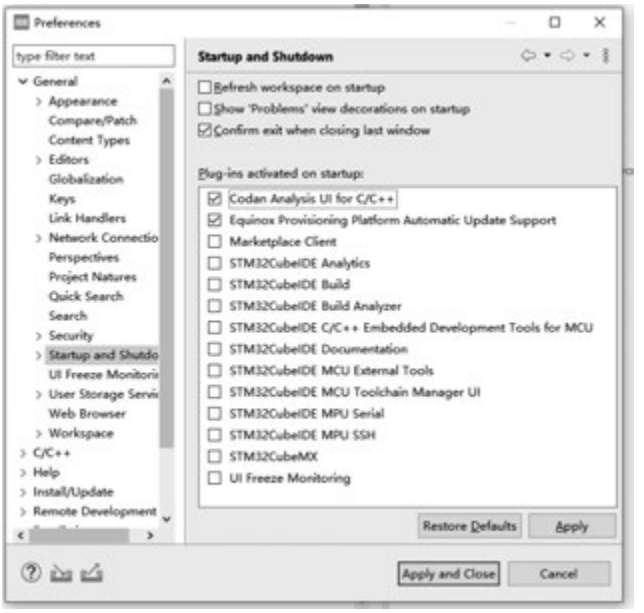

图 3-101　软件启动插件设置

图 3-102 所示为最近使用工作空间的设置，可以设置保存最近使用工作空间的个数、是否在软件启动时提示选择工作空间，也可以清空最近使用的工作空间列表。

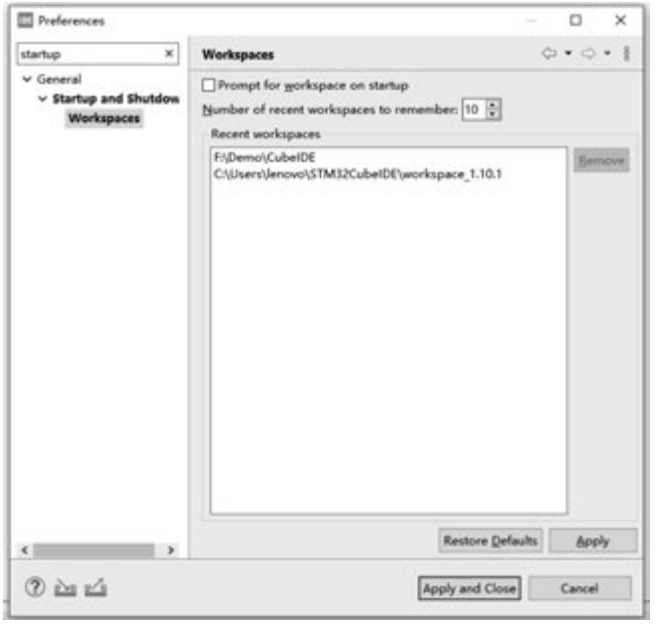

图 3-102　最近使用工作空间的设置

在文本框中输入 fold 进行搜索，进入如图 3-103 所示的代码折叠设置页面。图 3-103 中的第二个和第三个复选框默认是不勾选的，勾选这两个复选框对于查看和分析代码比较有用。

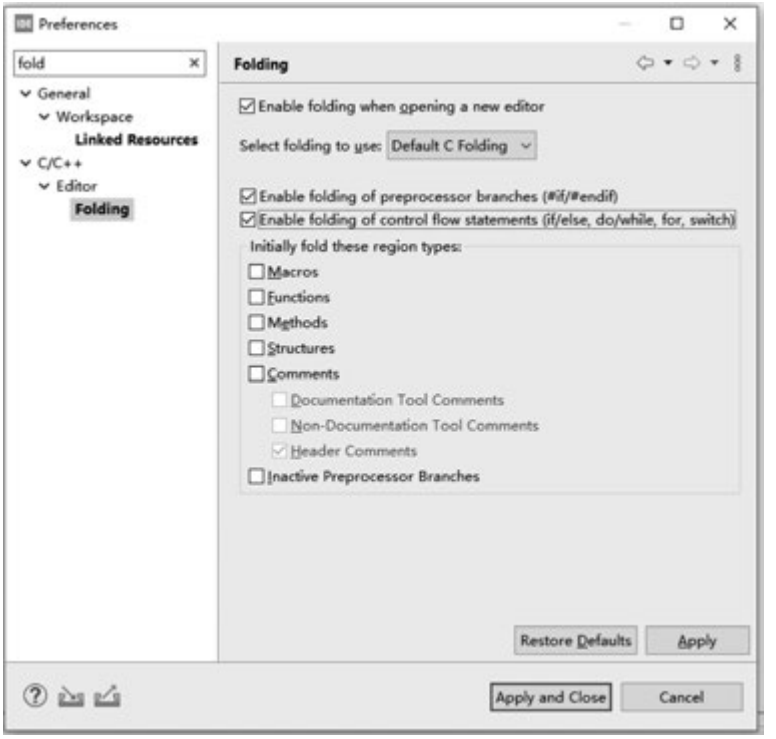

图 3-103 代码折叠设置页面

Preferences 对话框还有很多设置功能，主菜单里也还有很多功能，无法在此全部介绍，读者可以自己在使用中逐步摸索和发现。

3.9 STM32F407 开发板的选择

本书应用实例是在野火 F407-霸天虎开发板上调试通过的，该开发板可以网上购买，价格因模块配置的区别而不同，一般为 500～700 元。

野火 F407-霸天虎开发板使用 STM32F407ZGT6 作为主控芯片，使用 3.3 英寸液晶屏进行交互；可通过 Wi-Fi 形式接入互联网，支持使用串口（TTL）、RS-485、CAN、USB 协议与其他设备通信；板载 Flash、EEPROM、全彩 RGB LED 灯；还提供了各式通用接口，能满足各种各样的学习需求。

野火 F407-霸天虎开发板如图 3-104 所示。

图 3-104　野火 F407-霸天虎开发板(带 TFT LCD)

3.10　STM32 仿真器的选择

1. ST-LINK/V2 仿真器

ST-LINK 是 ST 公司为 STM8 和 STM32 系列微控制器设计的仿真器。ST-LINK/V2 仿真器如图 3-105 所示。

ST-LINK 仿真器具有如下特点。

(1) 编程功能：可烧写 Flash ROM、EEPROM 等，需要安装驱动程序才能使用。

(2) 仿真功能：支持全速运行、单步调试、断点调试等调试方法。

(3) 可查看 I/O 状态、变量数据等。

(4) 仿真性能：采用 USB 2.0 接口进行仿真调试、单步调试、断点调试，反应速度快。

(5) 编程性能：采用 USB 2.0 接口，进行 SWIM/JTAG/SWD 下载，下载速度快。

2. 微控制器调试接口

STM32 系列微控制器调试接口引脚图如图 3-106 所示。为了减少 PCB 的占用空间，JTAG 调试接口可用双排 10 引脚接口，SWD 调试接口只需要 SWDIO、SWCLK、RESET 和 GND 这 4 根线。

图 3-105　ST-LINK/V2 仿真器

图 3-106　STM32 系列微控制器调试接口引脚图