

# 项目简要开发文档

## 1. 项目名称

C++ Shell 课程设计

## 2. 开发环境

操作系统: Linux

编译器: g++

构建工具: CMake

依赖库: 标准 C++ 库, POSIX API

## 3. 项目简介

本项目实现了一个简单的 Shell 程序，能够解析并执行用户输入的命令，支持基本的 I/O 重定向和管道功能。用户可以在该 Shell 中输入命令来执行，直到输入 exit 命令退出 Shell。

### 3.1 开发状态

完成度: 核心功能已实现，包括命令解析、执行、I/O 重定向、管道处理。

质量: 程序基本稳定，经过初步测试，能够正确处理常见的 Shell 命令和操作。

### 3.2 核心内容

#### 1. 命令解析:

通过空格分割用户输入的命令字符串，支持多个命令通过管道符 | 连接。

支持输入重定向 < 和输出重定向 >。

#### 2. 命令执行:

使用 `fork()` 创建子进程执行命令。

使用 `execvp()` 替换子进程的执行映像为用户命令。

父进程等待子进程完成执行。

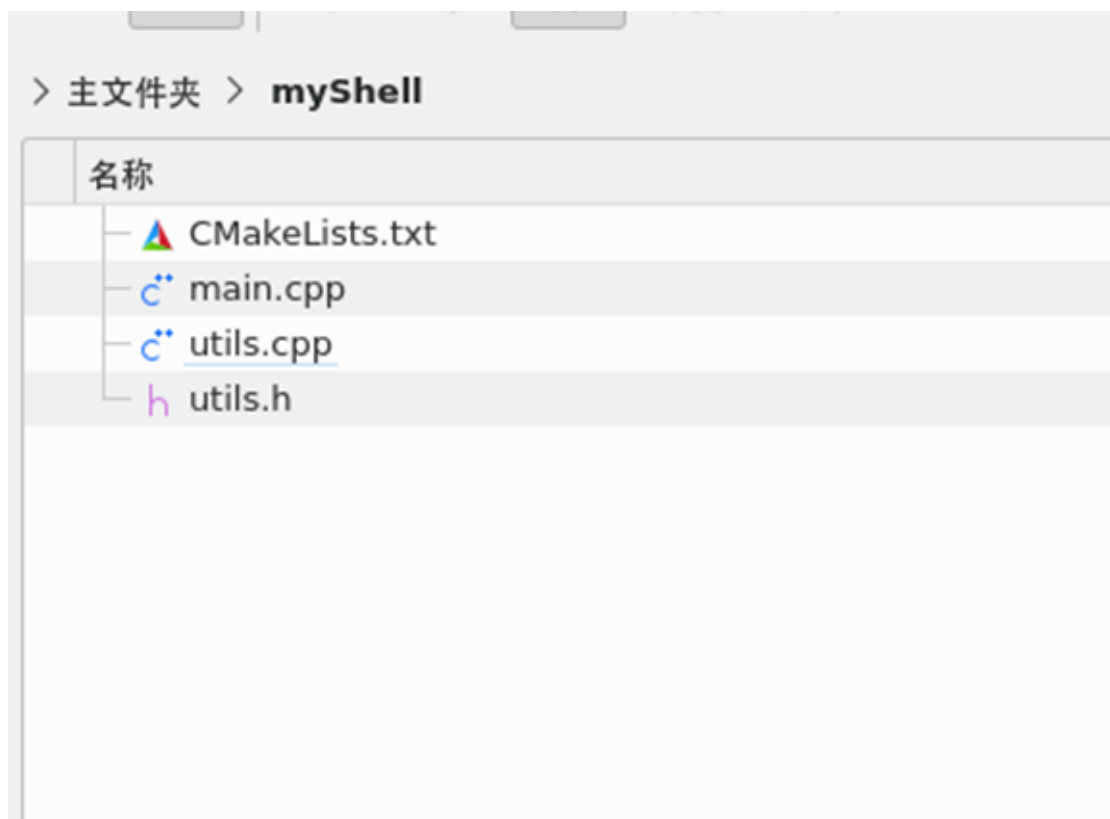
### 3. I/O 重定向:

子进程在执行命令前，通过 `dup2()` 重定向标准输入/输出到指定文件。

### 4. 管道处理:

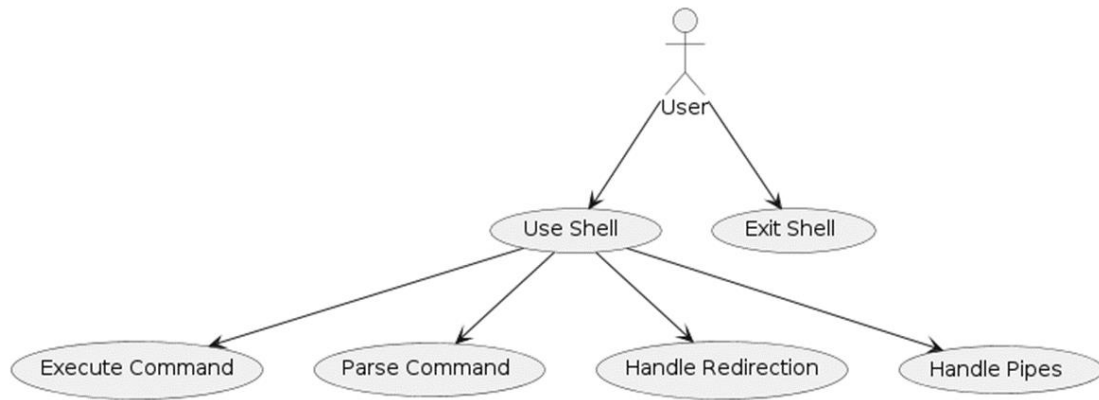
通过 `pipe()` 创建管道，将多个命令连接起来，使前一个命令的输出成为下一个命令的输入。

## 3.3 代码结构



## 3.4 uml 程序设计图

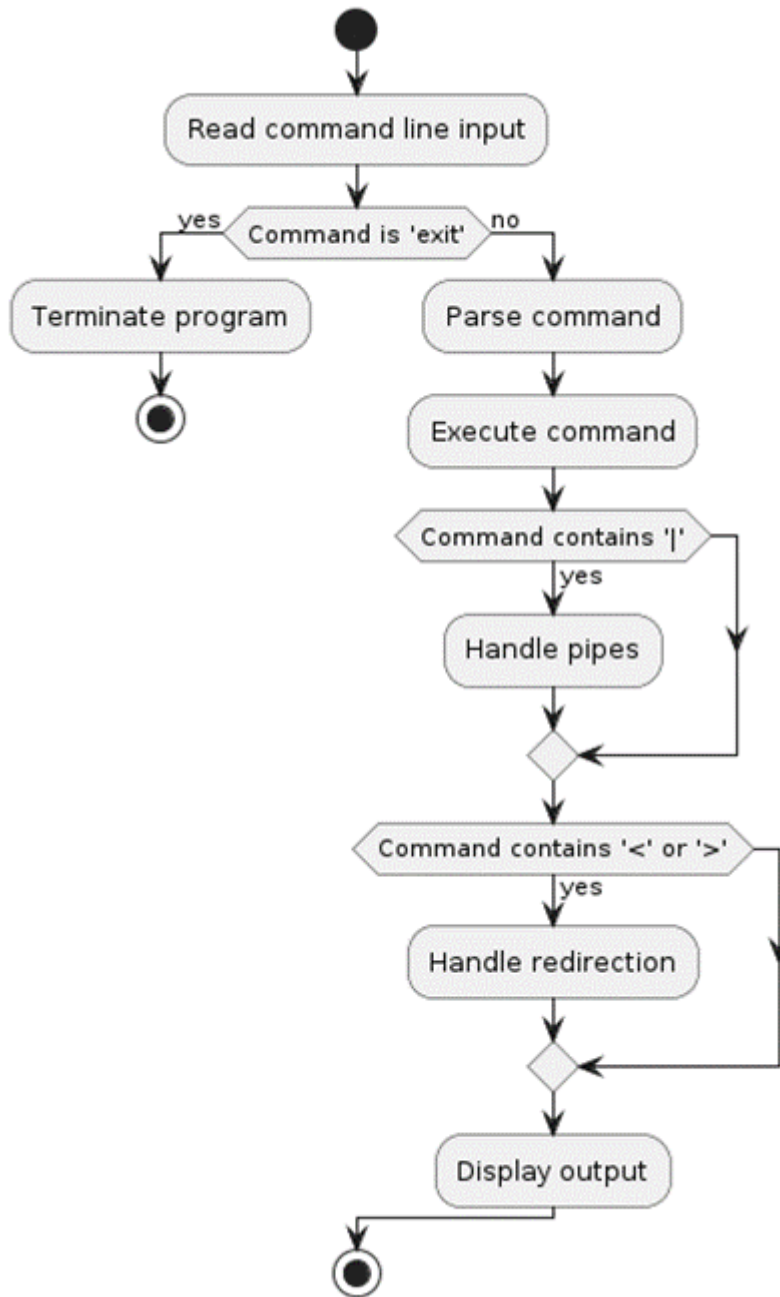
### 3.4.1 用例图:



- 用户(User)：主要的交互者，代表使用 Shell 的用户。
- 使用 Shell(Use Shell)：用户使用 Shell 的主要用例。
- 退出 Shell(Exit Shell)：用户退出 Shell 的操作。
- 解析命令(Parse Command)：Shell 解析用户输入的命令。
- 执行命令(Execute Command)：Shell 执行用户输入的命令。
- 处理重定向(Handle Redirection)：Shell 处理输入和输出重定向操作。
- 处理管道(Handle Pipes)：Shell 处理管道操作。

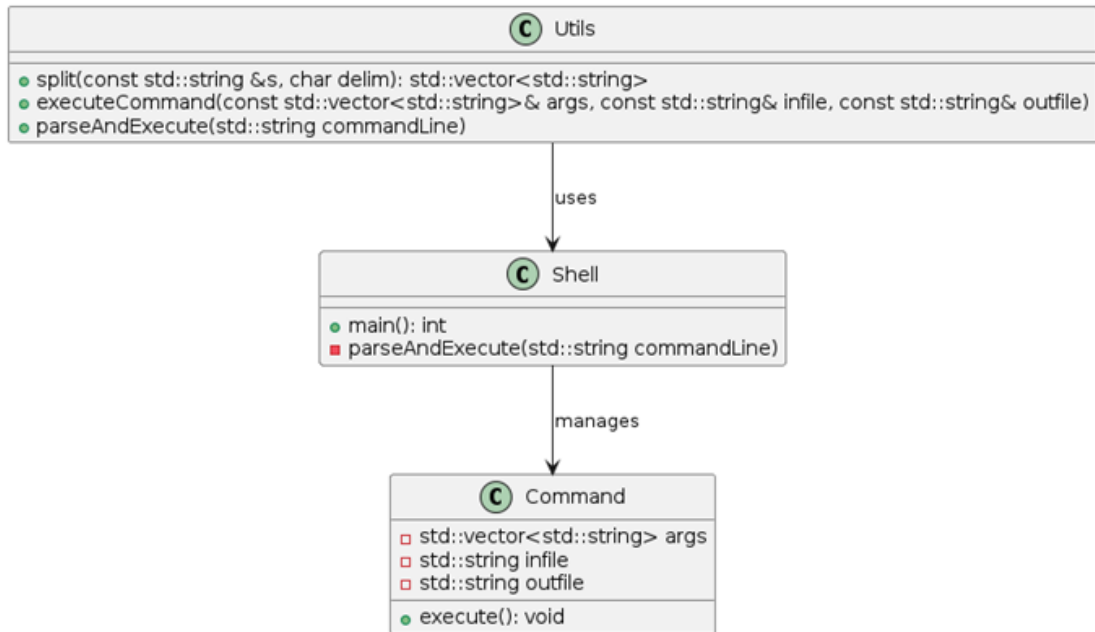
本用例图展示了用户在使用 Shell 时可能执行的主要操作，以及 Shell 在执行这些操作时涉及的主要功能，包括解析，执行命令，处理重定向和管道的功能。

#### 3.4.2 流程图



程序从开始运行时，首先读取用户输入的命令行。然后检查用户输入的命令是否为'exit'，如果是，则终止程序；如果不是，则解析用户输入的命令。解析完成后，执行命令，并根据命令是否包含管道符号('|')或重定向符号('<'或 '>')来相应处理管道或重定向操作。最后，显示命令执行的输出结果，程序运行结束。

#### 3.4.3 类图



**Utils 类:** 包含实用函数，用于字符串分割、命令执行以及命令解析。

- `split`: 将字符串按照指定分隔符分割成子字符串。
- `executeCommand`: 执行一个命令，并处理输入输出重定向。
- `parseAndExecute`: 解析命令行并执行。

**Shell 类:** 表示 Shell 程序的入口。

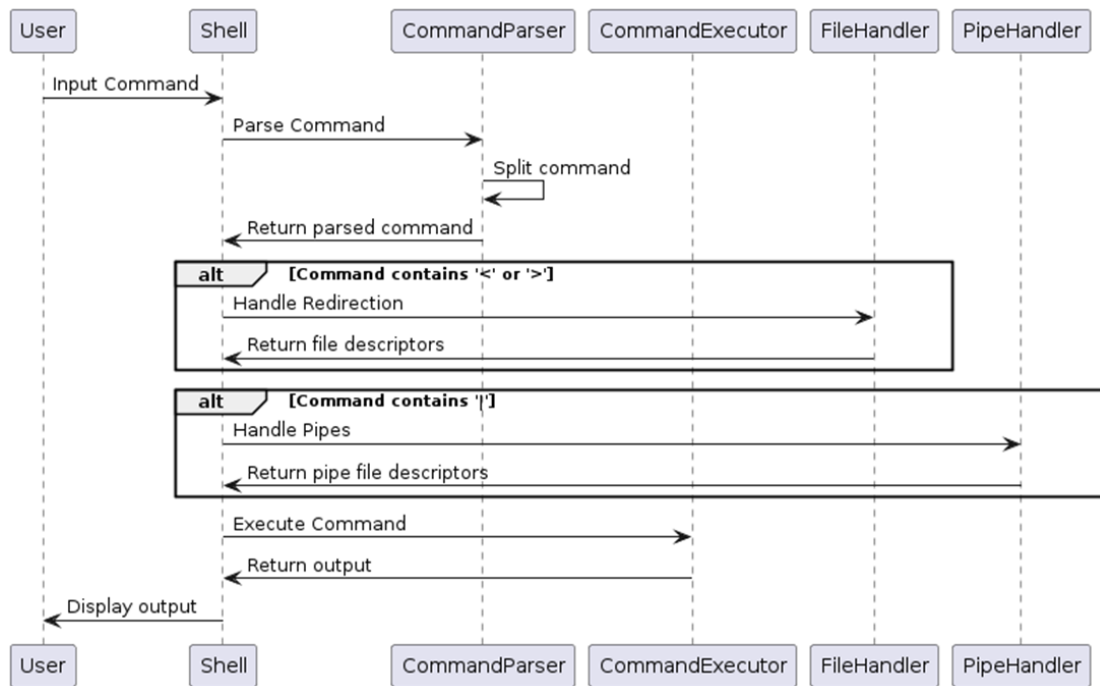
- `main`: 程序的主入口。
- `parseAndExecute`: 解析和执行命令行。

**Command 类:** 表示一个命令的详细信息。

- `args`: 命令的参数。
- `infile`: 输入重定向文件。
- `outfile`: 输出重定向文件。
- `execute`: 执行命令。

**关系:** Shell 类使用 Utils 类的方法，管理命令的执行。

#### 3.4.4 时序图



用户(User) 向 Shell 输入命令。Shell 调用 Utils 的方法来解析命令。Utils 返回解析后的命令给 Shell。根据命令内容：如果命令包含重定向符号，Shell 处理重定向，Command 返回文件描述符；如果命令包含管道符号，Shell 处理管道，Command 返回管道文件描述符。

然后 Shell 执行命令，Command 返回执行结果，Shell 将执行结果显示给用户(User)。

## 4. 项目难点与测试

### 4.1 技术重难点

1. 命令解析：需要正确处理命令中的特殊符号，如 |, <, >, 并正确分割命令和参数。
2. I/O 重定向：在子进程中使用 dup2() 函数实现标准输入输出的重定向。
3. 管道处理：在多个命令间正确创建和管理管道，以实现命令的流水线操作。
4. 子进程管理：使用 fork() 创建子进程，使用 execvp() 执行命令，并在父进程中使用 waitpid() 等待子进程完成。

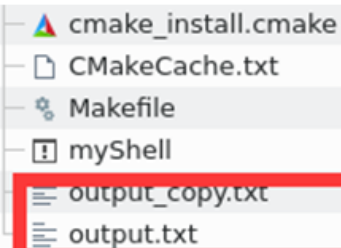
### 4.2 测试及结果展示

#### 1. 简单命令

```
root ~ > myShell > build > ./myShell
myshell> ls
CMakeCache.txt CMakeFiles cmake_install.cmake Makefile myShell
myshell> pwd
/root/myShell/build
myshell> echo "Hello,World"
"Hello,World"
myshell> date
2024年 06月 30日 星期日 19:11:15 CST
myshell> □
```

## 2. 带有重定向的命令

```
myshell> echo "Hello" > output.txt
myshell> cat output.txt
"Hello"
myshell> cat < output.txt > output copy.txt
```

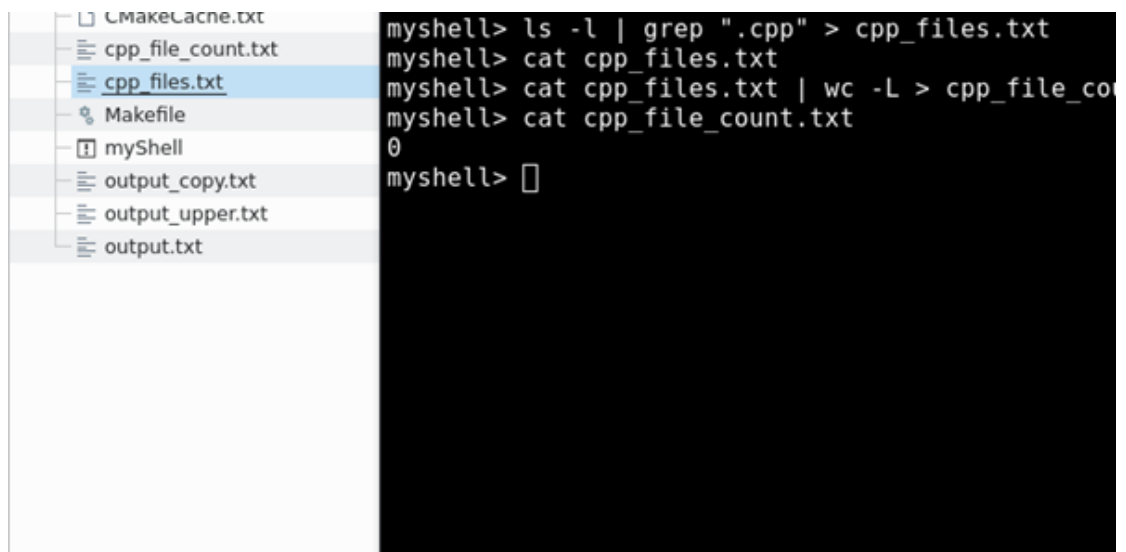


```
— cmake_install.cmake
— CMakeCache.txt
— Makefile
— myShell
— output_copy.txt
— output.txt
```

## 3. 管道命令

```
root ~ > myShell > build > ./
root ~ > myShell > build > ./myShell
myshell> ls | grep "utils"
myshell> echo "Hello" | tr '[:lower:]' '[:upper:]'
"HELLO"
myshell> cat < output.txt | tr '[:lower:]' '[:upper:]' > output_upper.txt
myshell> cat output_upper.txt
"HELLO"
myshell> □
```

## 4. 组合命令



## 5. 收获与反思

在本次 C++ Shell 项目的开发过程中，我获得了许多宝贵的经验和知识，具体体现在以下几个方面：

首先，通过实现 Shell 的命令解析和执行功能，我对 Shell 的工作机制有了深入的理解。之前只是在使用 Shell 命令，但这次开发让我真正了解了 Shell 如何解析用户输入、如何处理 I/O 重定向以及如何通过管道连接多个命令。特别是在命令解析方面，我学习了如何处理字符串分割、转义字符以及特殊符号等复杂情况，这对我以后处理文本解析问题提供了很好的借鉴。

其次，在开发过程中，我进一步掌握了 C++ 中的进程管理和进程间通信。使用 `fork()` 创建子进程，使用 `execvp()` 执行命令，并在父进程中使用 `waitpid()` 等待子进程完成，这些操作不仅加深了我对 POSIX API 的理解，也让我对操作系统的进程管理有了更直观的认识。此外，通过实现 I/O 重定向和管道功能，我学习了 `dup2()` 和 `pipe()` 的使用，这些技术将对我以后处理系统编程和网络编程中的类似问题有很大的帮助。

再次，在调试和解决问题的过程中，我的调试能力和解决问题的能力得到了提高。在开发过程中，我遇到了很多意想不到的问题，如处理字符串分割时的边界情况、进程间通信中的数据流管理等。通过不断地调试和查阅资料，我逐步解决了这些问题，积累了丰富的实践经验。这不仅提高了我的代码调试能力，也让我学会了如何在遇到问题时快速定位和解决问题。

总的来说，通过这个项目，我不仅学到了许多新的技术和知识，还提高了我的编程能力和问题解决能力，为我以后的开发工作打下了坚实的基础。