CMake 教程第三集：使用 CMake 变量及内部/外部构建

一、前言

在前两集中，我们介绍了 CMake 的基础命令和如何创建简单的CMake项目。本集将继续深入CMake的常用变量，尤其是 `<projectname>_SOURCE_DIR` 和 `<projectname>_BINARY_DIR`，并讨论 CMake 中的内部构建与外部构建的区别。

二、CMake 中的常用路径变量

CMake 自动定义了一些与项目路径相关的变量，帮助开发者更方便地管理源代码和构建目录。在这一部分，我们将重点讲解其中的两个关键变量。

1. `<projectname>_SOURCE_DIR`

- 该变量用于表示项目的源代码目录。
- **定义：**
  当你创建一个项目时，CMake 会自动为项目生成一个名为 `<projectname>_SOURCE_DIR` 的变量，表示源代码的顶级目录。

  ```
  # 示例：假设项目名为 MyProject
  message("Source Directory: ${MyProject_SOURCE_DIR}")
  ```

- **用途：**
  该变量通常用于引用源代码中的文件路径，特别是在项目有复杂的文件组织结构时，帮助我们在CMakeLists.txt文件中轻松定位源文件。

2. `<projectname>_BINARY_DIR`

- 该变量表示项目的构建目录（即生成二进制文件的目录）。

- **定义：**
  CMake 也会自动为项目生成一个名为 `<projectname>_BINARY_DIR` 的变量，表示生成的二进制文件（如编译的可执行文件或库文件）所在的顶级目录。

```
# 示例：假设项目名为 MyProject
message("Binary Directory: ${MyProject_BINARY_DIR}")
```

- **用途：**
  该变量用于引用编译生成的目标文件的存放路径，尤其在需要跨多个目录管理输出文件时非常有用。

3. `CMAKE_SOURCE_DIR` 和 `CMAKE_BINARY_DIR`

- 除了项目特定的变量，CMake 还提供了全局变量 `CMAKE_SOURCE_DIR` 和 `CMAKE_BINARY_DIR`，它们分别指代当前CMake项目的源代码目录和构建目录。

```
message("Global Source Directory: ${CMAKE_SOURCE_DIR}")
message("Global Binary Directory: ${CMAKE_BINARY_DIR}")
```

- **区别：**
  - 如果你只有一个顶层项目，这些全局变量和项目相关的变量（如 `<projectname>_SOURCE_DIR`）是相同的。
  - 在多项目构建环境中，项目相关的变量指向具体项目的源代码目录或构建目录，而全局变量指向顶层目录。

三、内部构建 vs 外部构建

在CMake中，构建方式主要有两种：**内部构建**和**外部构建**。这两种方式各有优劣，适用于不同的项目需求。

1. 内部构建 (In-Source Build)

- **定义：**
  内部构建是指构建过程发生在源代码目录内，即CMake生成的构建文件（如Makefile、编译器缓存等）直接存储在项目的源代码目录中。
  - **优点：**

- 简单直接，特别适用于小型项目。

- **缺点：**
  - 源代码和构建文件混在一起，容易导致文件夹混乱。
  - 重新构建时需要清理源代码目录，容易误删源文件。
- **如何执行内部构建：**

```
cmake .
make
```

2. 外部构建 (Out-of-Source Build)

- **定义：**
  外部构建是指将构建过程的所有输出文件存储在一个单独的目录中，而不是源代码目录内。推荐在大多数情况下使用这种方式。
  - **优点：**
    - 源代码目录保持干净，不会被生成的临时文件污染。
    - 可以在不同的构建目录中针对不同的编译配置（如Debug和Release）进行独立构建。
  - **缺点：**
    - 需要管理多个文件夹，可能会增加复杂度。
  - **如何执行外部构建：**
    首先在项目目录外创建一个 `build` 文件夹，然后在 `build` 文件夹内运行CMake构建命令。

```
mkdir build
cd build
cmake ..
make
```

    构建完成后，所有生成的文件（如可执行文件、编译中间文件等）都会存放在 `build` 目录中，源代码目录不会被影响。

四、小结

本集我们介绍了如何使用 CMake 中的路径相关变量，包括 `<projectname>_SOURCE_DIR` 和 `<projectname>_BINARY_DIR`，以及全局路径变量 `CMAKE_SOURCE_DIR` 和 `CMAKE_BINARY_DIR`。此外，我们还讨论了内部构建和外部构建的区别，并推荐在大多数情况下使用外部构建以保持项目结构清晰。

在下一集中，我们将继续深入探讨CMake的更多进阶特性，包括如何处理库文件和外部依赖管理。

五、作业

1. 使用你自己的项目，在项目目录下同时执行一次内部构建和外部构建，观察两者的区别。
2. 使用 `message()` 打印项目的源代码目录和构建目录，尝试在 CMakeLists.txt 文件中引用这些变量。