CMake Learning

- author: zhouyongsdzh@foxmail.com

- date: 20160516

0. 关于cmake

0.1. cmake综述

历史 && 功能 (常规与先进) && 应用场景

0.2. 一个简单的cmake示例

项目目录如下：

```
/home/zhouyong/myhome/2016-Planning/C-CPP/cmake-learning/cmake_1_samples
```

目录下两个文件，分别是：main.cc和**CMakeLists.txt** 内容分别为：

main.cpp文件

```
/*
 * File Name: main.cpp
 * Author: zhouyongsdzh@foxmail.com
 * Created Time: 2016-08-18 22:57:11
 */

#include <iostream>
using namespace std;

int main() {
    std::cout << "cmake samples! come on." << std::endl;
    return 0;
}
```

一个简单的输出语句。**CMakeLists.txt**内容为：

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.2)

####
# 1. PROJECT NAME

PROJECT(cmake_samples)

####
# 2. set && initialize info

SET(SRC_LISTS main.cpp)         # 设置. SET关键词功能为赋值。这里即：SRC_LISTS=main.cpp

MESSAGE("[INFO] binary dir: ${PROJECT_BINARY_DIR}")
MESSAGE("[INFO] source dir: ${PROJECT_SOURCE_DIR}")
MESSAGE("[INFO] src_lists: ${SRC_LISTS}")   # 输出内容为 main.cpp
####
# 3. executable

ADD_EXECUTABLE(cmake_samples_exec ${SRC_LISTS})
```

然后，在当前目录下建立build构建目录 (外部构建，out-of-source build)，进入，分别执行:

```
mkdir build
cd build
cmake ..            # CMakeLists.txt文件所在目录
make                # 构建，输出可执行文件
```

这样在build目录下可以看到一个名为 `cmake_samples_exec` 的可执行文件. 运行可得:

```
zhouyong@ubuntu:~/myhome/2016-Planning/C-CPP/cmake-learning/cmake_1_samples/build$ ./cmake_sam
cmake samples! come on.    # 输出结果.
```

至此，一个简单的cmake构建项目就完成了。

0.3. 先睹为快 (cmake语法)

在0.2中通过一个简单的示例，介绍了cmake在项目构建中时如何使用的。CMakeLists.txt文件中一些语法 先简单介绍一下。先说明一点：**cmake关键词不区分大小写，大小写含义相同**.

> 注意：这里说的是关键词不分大小写，而不是**cmake**变量。谨记！！！

关键词	含义
`cmake_minimum_required(version 3.2)`	指定cmake最低版本。这里是3.2版.
`PROJECT(project_name)`	project关键字用于指定项目名称.
`SET(var value)`	设置。将value赋给变量var.
`MESSAGE(str_info)`	打印信息
`PROJECT_BINARY_DIR`	标识目标二进制文件目录。
`PROJECT_SOURCE_DIR`	标识目标源文件目录。
`ADD_EXECUTABLE`	添加可执行文件名称，以及所需要的源文件。

2. cmake语法

2.1. cmake指令

如何使用外部头文件和共享库?

- `INCLUDE_DIRECTORIES`

完整语法: `INCLUDE_DIRECTORIES([AFTER|BEFORE] [SYSTEM] dir1 dir2 ...)`

2.2. cmake常用的变量 (不是关键词，注意大小写)

cmake使用 `${}` 进行变量的饮用。在IF等语句中，直接使用变量名而不通过${}取值。

cmake自定义变量主要有隐式定义和显士定义两种。PROJECT指令就属于隐式定义，它会间接的定义 `<project_name>_BINARY_DIR` 和 `<project_name>_SOURCE_DIR` 两个变量。

显式定义 使用SET指令，可以构建一个自定义变量。

变量名	含义
`CMAKE_BINARY_DIR` `PROJECT_BINARY_DIR`	如果是in-source编译，指的是工程顶层目录；如果是out-of-source编译，指的是工程编译发生的目录
`CMAKE_SOURCE_DIR` `PROJECT_SOURCE_DIR`	不管是in-source编译还是out-of-source编译，都是指工程顶层目录
`CMAKE_CURRENT_SOURCE_DIR`	指的是当前处理CMakeLists.txt所在的路径. 如果是src下面的CMakeLists.txt, 则该值为 `${project_source_dir}/src`
`CMAKE_CURRENT_BINARY_DIR`	如果是in-source编译，它与 `CMAKE_CURRENT_SOURCE_DIR` 一致，如果是out-of-source编译，它指的是target编译目录，该值为 `${project_source_dir}/build/bin`
`CMAKE_CURRENT_LIST_FILE`	当前CMakeList.txt文件对应的完整路径
`CMAKE_CURRENT_LIST_LINE`	输出CMakeList.txt中该变量所在的行
`CMAKE_MODULE_PATH`	这个变量用来定义自己的cmake模块所在的路径。工程较复杂时，可能需要编写cmake模块，需要用SET指令，将自己的cmake模块路径设置一下。比如 `SET(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake)` , 然后通过include指令 调用自己的模块了
`EXECUTABLE_OUTPUT_PATH`	指定目标二进制的位置（最终生成的exec）。不论是否制定编译输出目录（subdirs或 `add_subdirector` 指令）, 都可以使用set指令重新设置。 `SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)`
`LIBRARY_OUTPUT_PATH`	指定目标二进制的位置（最终的共享库）。不论是否制定编译输出目录（subdirs或 `add_subdirectory` 指令）, 都可以使用set指令重新设置。 `SET(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)`
`PROJECT_NAME`	返回通过project指令定义的项目名称

2.3. cmake常用的环境变量

使用 `$ENV{NAME}` 指令调用系统的环境变量。例如： `MESSAGE(STATUS "home dir: $EMV{HOME}")` .

设置环境变量的方式： `SET (ENV{变量名} 值)`

常用的环境变量 如下:

环境变量名	含义
`CMAKE_INCLUDE_CURRENT_DIR`	

2.4. cmake系统信息

1. 使用cmake驾驭工程

一个c/cpp项目工程的结构一般是什么样的？如何加入第三方（third party）工具作为当前项目的引用?

1.1. c/cpp项目工程

一个常见的c/cpp项目工程如下：

```
```